

**FANSYS: A Computer Model of
Text Comprehension and Question Answering
for Failure Analysis***

**Sergio J. Alvarado
Ronald K. Braun
Kenrick J. Mock**

Technical Report CSE-93-4

INFORMATION SCIENCES LIBRARY
AMES RESEARCH CENTER
MOFFETT FIELD, CALIF.

JUL 15 1986

*Funds for the support of this study have been allocated by the NASA-Ames Research Center, Moffett Field, California, under Interchange No. NCA2-721.

Table of Contents

Contents	Page
List of Figures	iv
Abstract	vi
1. Introduction	1
2. Issues Addressed in FANSYS.....	2
3. System Architecture	5
3.1 Overview of System Modules	5
3.2 Indexing Hierarchies	5
4. Domain Knowledge Representation.....	13
4.1 The DMS Fault Domain.....	13
4.1.1 DMS Entities.....	14
4.1.2 DMS States.....	16
4.1.3 DMS Actions	19
4.1.4 DMS Events.....	21
4.1.5 DMS Procedures	22
4.2 Case Representation.....	28
5. Comprehension of Input Text and Questions	36
5.1 Case-Based Parsing.....	39
5.2 Parsing Input Text	41
5.2.1 Marker Passing.....	43
5.2.2 The Parsing Algorithm.....	45
5.2.3 Domain Specific Inference Strategies.....	47
5.2.4 An Annotated Example.....	49
5.3 Parsing Input Questions	76
5.3.1 Question Patterns	77
5.3.2 An Annotated Question Example.....	78
5.4 Generating Answers to Input Questions	80
6. Memory Creation and Retrieval.....	84
6.1 MOP Attributes	84
6.2 Linking MOPs	86
6.3 Comparing Cases	88
6.4 Memory Creation Process	93
6.5 Memory Retrieval Process	98
7. User Interface.....	109
8. Current Status	125
9. Future Work	126
9.1 User Interface	126
9.2 Integration with Other Tools.....	126
9.3 Integration of Memory Structures	126
9.4 Implementation Scope.....	127
9.5 Knowledge Acquisition	127
9.6 Model and Functional Based Reasoning	128
9.7 Subjective Comprehension of Input Text.....	128
9.8 Belief Inferences Based on Past Failure Analysis	129

10. Conclusion	130
11. References.....	131
Appendix A: Case Descriptions	
Appendix B: Detailed I/O Examples	
Appendix C: Code Listings	

List of Figures

Figure Title	Page
Figure 2.1 Input Failure Description : Ring Concentrator Case 1 (RC.1).....	3
Figure 2.2 Question Answering Session for Ring Concentrator Case 1 (RC.1).....	4
Figure 3.1 FANSYS System Modules.....	6
Figure 3.2 FANSYS Flow of Control	7
Figure 3.3 FANSYS Link Types for the Power-Up Event.....	9
Figure 3.4 Sample Abstraction Hierarchy - Indexing of Concepts within Memory	11
Figure 3.5 Partial Memory Abstraction Hierarchy	12
Figure 4.1 Knowledge Representation in FANSYS.....	13
Figure 4.2 DMS Fault Domain Classes	14
Figure 4.3 DMS Entities.....	15
Figure 4.4 Entities: Memory Organization Example.....	16
Figure 4.5 Entities: Textual Mapping Example	17
Figure 4.6 DMS States	18
Figure 4.7 States: Memory Organization Example	18
Figure 4.8 States: Textual Mapping Example.....	19
Figure 4.9 DMS Actions	20
Figure 4.10 Actions: Memory Organization Example.....	20
Figure 4.11 Actions: Instance Example.....	21
Figure 4.12 DMS Events	21
Figure 4.13 Events: Memory Organization Example	23
Figure 4.14 Events: Textual Mapping Example.....	24
Figure 4.15 DMS Procedures	25
Figure 4.16 Procedures: Memory Organization Example	26
Figure 4.17 Procedures: Textual Mapping Example	27
Figure 4.18 Representing a Case in FANSYS	28
Figure 4.19 Failed Component Mapping for RC.1	29
Figure 4.20 Failure Mode Mapping for RC.1	29
Figure 4.21 Failure Cause Mapping for RC.1.....	29
Figure 4.22 Failure Detection Mapping for RC.1	30
Figure 4.23 Failure Correction Mapping for RC.1 (Short Term)	31
Figure 4.24 Failure Correction Mapping for RC.1 (Long Term).....	33
Figure 4.25 Failure Effects Mapping for RC.1	34
Figure 4.26 Case Representation for RC.1.....	35
Figure 5.1 Example Mapping of Text for the Power-Up Event	36
Figure 5.2 Index Pattern Structure.....	43
Figure 5.3 Activation Marker Structure	44
Figure 5.4 Prediction Marker Structure	44
Figure 5.5 The Two Question Formats	78
Figure 5.6 Sample Hierarchy for I-M-Time-Out-Event.144.507	82
Figure 6.1 Sample GI Memory Hierarchy	87
Figure 6.2 Graphical Representation of Sample Query	90
Figure 6.3 Graphical Representation of Sample Case	90
Figure 6.4 Comparing Case and Query	92

Figure 6.5 GI Hierarchy After Adding RC Case to Memory.....	96
Figure 6.6 GI Hierarchy After Adding RC and GW Case to Memory.....	97
Figure 6.7 Traversal Paths for Example Query - Direct Retrieval	101
Figure 6.8 Traversal Paths for Example Query - Alternate Entry MOP	103
Figure 6.9 Traversal Paths for Example Query with Elaboration	106
Figure 7.1 FANSYS User Interface	110
Figure 7.2 Trace Window: Parsing RC.1	111
Figure 7.3 Query Construction via Menu: Selection of Requested Items	113
Figure 7.4 Query Construction via Menu: Selection of Failed Component.....	114
Figure 7.5 Query Construction via Menu: Selection of Given Information	115
Figure 7.6 Query Construction via Menu: Selection of Failure Mode	116
Figure 7.7 Query Construction via Menu: Selection of Failure Cause.....	117
Figure 7.8 Query Construction via Menu: Selection of Failure Detection	118
Figure 7.9 Query Construction via Menu: Selection of Failure Correction.....	119
Figure 7.10 FANSYS Help Window Showing Query Format.....	121
Figure 7.11 Sample of the Question Shell	122
Figure 7.12 Memory Inspector: Expansion of Ring Concentrator Case 1	124

Abstract

The research described in this technical report is aimed at extending previous theories of natural language processing and memory search and retrieval. In particular, this research has been concerned with the development of a model of text comprehension and question answering that uses case-based reasoning techniques to acquire knowledge of failure analysis from input text, and answer questions regarding diagnosis and repair of failures in complex systems, such as NASA's Space Station Freedom. The major goal has been to develop a process model that accounts for (a) how to build a knowledge base of system failures and repair procedures from textual descriptions that appear in NASA's FMEA (failure modes and effects analysis) manuals, and (b) how to use that knowledge base to evaluate causes and effects of failures, and provide diagnosis and repair procedures. This process model has been implemented in an experimental computer program called *FANSYS* (Failure ANalysis SYStem), which is a text comprehension and question answering system for the analysis of failures occurring within the data management system (DMS) of NASA's Space Station Freedom. This project received funds from NASA Ames Research Center to support three major tasks in a period of a year: (1) analysis of conceptual content of system-failure manuals; (2) characterization of the knowledge structures and processes underlying the computational analysis of failures; and (3) implementation of an experimental version of *FANSYS* that reads a few segments of the manuals, and answers a number of questions involving failure diagnosis and repair.

1. Introduction

A goal of artificial intelligence is to develop computer systems that exhibit skills similar to those used by humans during language comprehension, language generation, planning, reasoning, and argumentation (Alvarado, 1992). In the case of intelligent systems designed to aid in complex decision making and the generation of expert advice, it is necessary that they be able to evaluate given goal situations, present their beliefs on possible plans of action, justify their beliefs, understand opposing beliefs, and argue for/against given plans. These skills are needed because it is unlikely that any individual seeking expert advice would unquestioningly trust or follow the suggestions of systems that can propose a course of action, but cannot justify such proposals, and cannot understand or counter those objections that may be raised to their suggestions.

A step towards modeling the interrelationships that exist among reasoning, planning, and language comprehension has been taken by Alvarado (1990) with the development of a prototype editorial comprehension and question answering system called *OpEd*. In this system, the process of text comprehension is viewed as one of managing many different knowledge sources, such as causal chains of reasoning, goals, plans, actions, characters, beliefs, belief relationships, and argument units. Associated with each knowledge construct are one or more processing strategies that are invoked by a conceptual parser (Dyer, 1983) in order to build a network of beliefs (Flowers et al., 1982) that represents the content of the input text. During question answering, the system uses a question-categorization scheme (Lehnert, 1978) and retrieval heuristics (Dyer and Lehnert, 1982) to analyze each input question and select specific strategies of search and retrieval.

The research described in this technical report is aimed at extending the theories of natural language processing and memory search and retrieval underlying OpEd's design. In particular, this research has been concerned with the development of a model of text comprehension and question answering that uses case-based reasoning techniques (Riesbeck and Schank, 1989) to acquire knowledge of failure analysis from input text, and answer questions regarding diagnosis and repair of failures in complex systems, such as NASA's Space Station Freedom. The major goal has been to develop a process model that accounts for (a) how to build a knowledge base of system failures and repair procedures from textual descriptions that appear in NASA's FMEA (failure modes and effects analysis) manuals, and (b) how to use that knowledge base to evaluate causes and effects of failures, and provide diagnosis and repair procedures. This process model has been implemented in an experimental computer program called *FANSYS* (Failure ANalysis SYStem), which is a text comprehension and question answering system for the analysis of failures occurring within the data management system (DMS) of NASA's Space Station Freedom*. This project received funds from NASA Ames Research Center to support three major tasks in a period of a year: (1) analysis of conceptual content of system-failure manuals; (2) characterization of the knowledge structures and processes underlying the computational analysis of failures; and (3) implementation of an experimental version of *FANSYS* that reads a few segments of the manuals, and answers a number of questions involving failure diagnosis and repair.

*See Barlett et. al (1992), McDonnell Douglas (July, 1990), and IBM (July, 1990) for complete descriptions of the DMS.

2. Issues Addressed in FANSYS

The problem of analyzing and diagnosing failures that occur in mechanical or electronic devices has been addressed by other researchers in artificial intelligence and computer science. Some of their work has been concerned with the development of truth-maintenance systems (de Kleer and Williams, 1987; Struss, 1988), expert systems (Reed and Johnson, 1990), connectionist expert systems (Liu et al., 1991), case-based reasoning systems (Hammond and Hurwitz, 1988), and digraph-based analysis systems (Iverson and Patterson-Hine, 1990; Patterson-Hine and Iverson, 1990; Stevenson, Miller, and Austin, 1991). However, these researchers have not addressed the task of comprehension of natural language in descriptions of failures, which requires dynamically constructing a knowledge base of the failure cases from textual input. In contrast, the FANSYS project has been concerned with the domain knowledge, conceptual representations, natural language processing strategies, and search and retrieval strategies involved in understanding textual descriptions of failure cases and answering questions about such cases. FANSYS involves the use of techniques for parsing input failure descriptions into a network of cases that maintains the context for subsequent question answering.

Text comprehension and question answering in FANSYS involve four major tasks: (1) applying domain-specific knowledge (i.e., DMS fault domain); (2) mapping input text into conceptual structures that compose the internal representation of failure-analysis cases; (3) representing and indexing failure-analysis cases in memory by creating a library of cases; and (4) using the library of failure-analysis cases to answer questions regarding failure causes and effects, as well as failure detection and correction procedures. Input failure descriptions correspond to segments of FMEA manuals, and contain the essential wording of the original descriptions. Here "essential" means that the original descriptions have been edited to remove those parts which involve addressing issues that fall outside the scope of FANSYS (e.g., references to specific part numbers, references to supporting documentation, and descriptions of functions of DMS components). Figures 2.1 and 2.2 illustrate the current input/output behavior of FANSYS. The input failure case shown in Figure 2.1 is a fragment of a failure description appearing in the FMEA document number MDC H4563A (McDonnell Douglas, March 1990). A question answering session using this case is shown in figure 2.2. The computational resources needed to support this type of input/output behavior will be described in the rest of this technical report.

ITEM NAME: Ring Concentrator (RC)

FAILURE MODE: Loss of output - Failure to Start

FAILURE CAUSES: Piece-part failures, Contamination, Temperature (High or Low), Mechanical Shock, Thermal Shock

FAILURE DETECTION/VERIFICATION: Indication of a RC "failure to start" is first detected by the "next" active node on the network. Local System Management within the "next" node will reach a time-out limit for receipt of the network token.

CORRECTIVE ACTION:

- (A) Short Term: Network reconfiguration is effected automatically. The DMS network remains in operation in a reconfigured state with the failed RC bypassed.
- (B) Long Term: The crewmen check for "applied power" and check for "connector tightness". If the RC cannot then be placed in operation, it is removed and replaced with an ORU logistics spare.

FAILURE EFFECT ON:

- (A) Crew/SSPE: None. The conditions associated with the replacement of a RC within the network configuration are such that the network is already in a reconfigured state supplying full services.
- (B) Mission Support: None.
- (C) System: None.
- (D) Interfaces: None.

Figure 2.1
Input failure description : Ring Concentrator Case 1 (RC.1)

**WHAT IS THE FAILURE MODE FOR THE RING CONCENTRATOR?
LOSS OF OUTPUT - FAILURE TO START.**

**WHAT IS THE FAILURE CAUSE FOR THE RING CONCENTRATOR WHEN THE FAILURE MODE
IS LOSS OF OUTPUT - FAILURE TO START?**

- [1] PIECE-PART FAILURES.
- [2] CONTAMINATION.
- [3] TEMPERATURE (HIGH OR LOW).
- [4] MECHANICAL SHOCK.
- [5] THERMAL SHOCK.

**WHAT IS THE FAILURE EFFECT WHEN THE FAILED COMPONENT IS THE RING
CONCENTRATOR?**

- [1] FAILURE EFFECT ON THE CREW: SHORT TERM CORRECTION PROCEDURE: THE LOCAL SYSTEM MANAGEMENT LOGICALLY DISCONNECTS THE RING CONCENTRATOR FROM THE DMS NETWORK.
- [2] FAILURE EFFECT ON THE MISSION SUPPORT: THERE IS NO EFFECT.
- [3] FAILURE EFFECT ON THE SYSTEM: THERE IS NO EFFECT.
- [4] FAILURE EFFECT ON THE INTERFACES: THERE IS NO EFFECT.

**WHAT IS THE FAILURE DETECTION PROCEDURE WHEN THE FAILED COMPONENT IS THE
RING CONCENTRATOR AND THE FAILURE MODE IS LOSS OF OUTPUT - FAILURE TO START?
DETECTION PROCEDURE: THE DMS WAITS FOR A NETWORK TOKEN FROM THE RING
CONCENTRATOR. THE DMS WILL REACH A TIME-OUT LIMIT FOR RECEIPT OF THE NETWORK
TOKEN.**

**WHAT IS THE FAILURE CORRECTION PROCEDURE WHEN THE FAILED COMPONENT IS THE
RING CONCENTRATOR AND THE FAILURE MODE IS LOSS OF OUTPUT - FAILURE TO START?**

- [1] SHORT TERM CORRECTION PROCEDURE: THE SYSTEM MANAGEMENT LOGICALLY DISCONNECTS THE RING CONCENTRATOR FROM THE DMS NETWORK.
- [2] LONG TERM CORRECTION PROCEDURE: THE CREW CHECKS FOR "APPLIED POWER". THE CREW CHECKS FOR "CONNECTOR TIGHTNESS".
CORRECTION PROCEDURE: THE CREW PHYSICALLY DISCONNECTS THE RING CONCENTRATOR FROM THE DMS NETWORK. THE CREW PHYSICALLY CONNECTS THE BACKUP RING CONCENTRATOR TO THE DMS NETWORK. THE CREW LOGICALLY DISCONNECTS THE RING CONCENTRATOR FROM THE DMS NETWORK. THE CREW LOGICALLY CONNECTS THE BACKUP RING CONCENTRATOR TO THE DMS NETWORK.

Figure 2.2
Question Answering Session for Ring Concentrator Case 1 (RC.1)

3. System Architecture

At a high level, FANSYS may be decomposed into five major modules: *user interface*, *case-based parser*, *memory update and retrieval*, *main memory*, and the *natural language generator*. An overview of these modules and their interactions is presented below and in figure 3.1.

3.1 Overview of System Modules

User interaction with FANSYS is handled through the user interface module. The interface has been designed to ease user tasks by accepting both textual and menu-driven input. The case-based parsing module is responsible for reading input case descriptions and questions, and building an internal memory representation for what has been read. The memory update and retrieval module determines where cases should be stored in memory and how those cases will be retrieved during question answering. Once cases have been retrieved during question answering, the natural language generator converts the internal representation into English text.

Although the design of FANSYS involves a modular architecture, its modules work in a highly interactive manner. This interaction is illustrated by the flow of control diagram shown in figure 3.2. This diagram indicates that during knowledge acquisition, control flows from the user interface to the parser, and then to the memory module as each case description is parsed and indexed in memory. The diagram also shows that, during question answering, control flows either (1) from the user interface to the parser and then to the retrieval and generation modules, or (2) from the user interface to the retrieval and generation modules. Furthermore, the diagram indicates that both phases of processing in FANSYS require repeated access to main memory to store or retrieve cases.

Figure 3.2 also shows that user interaction with FANSYS occurs primarily during the question answering phase. The user interface supports a variety of options, including the ability to browse through the representations created in memory, obtain help or system information, and ask the system questions about what it has read via natural language or point-and-click buttons. If a question is given in natural language, then the system will parse the question into a memory search query. Conversely, the user may construct the search query by clicking on buttons which represent known objects or procedures. Once a search query has been created, the memory retrieval module will search memory for cases which answer the query. These cases will then be output in English by the natural language generator. If desired, the user may instruct the system to search for additional cases that may be relevant to the search query.

3.2 Indexing Hierarchies

The most elaborate components of FANSYS are the parsing module, update and retrieval module, and the indices which both of these modules use to access memory. Both modules use a set of hierarchical indices to access and organize memory. Memory is organized in terms of Memory Organization Packets (MOP) (Schank, 1982; Riesbeck and Schank, 1989), where each MOP represents a concept. Each MOP is a frame (Charniak, 1977, 1978; Minsky, 1975, 1977) of slots and fillers, where the slots and fillers are defining attributes of the concept. The most

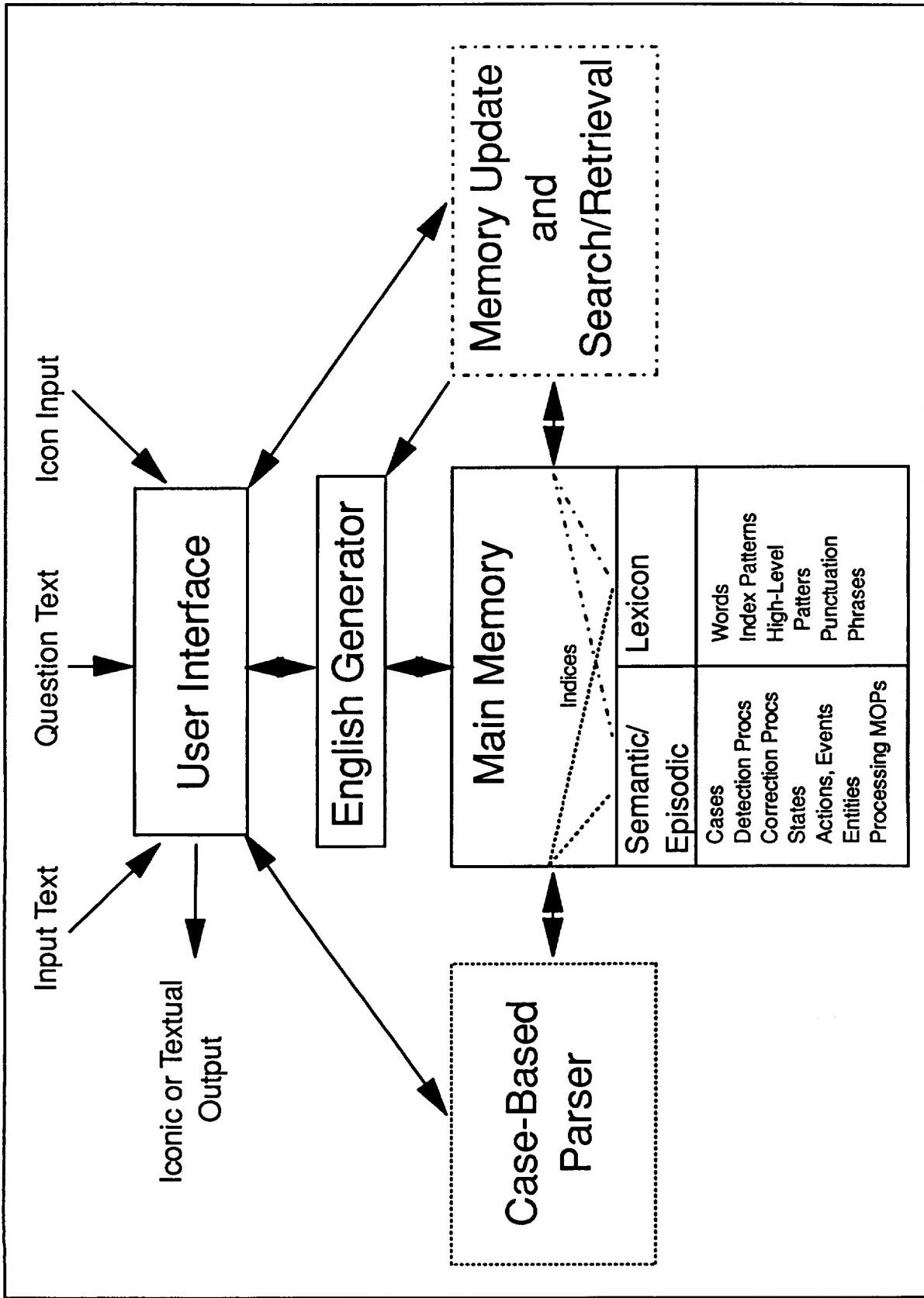
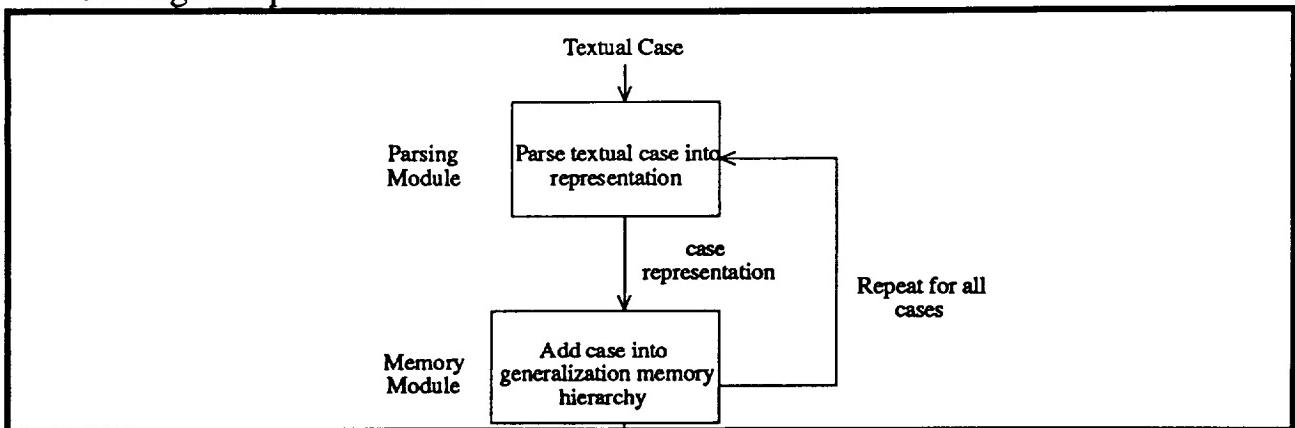


Figure 3.1
FANSYS System Modules

Knowledge Acquisition Phase



Question/Answer Phase

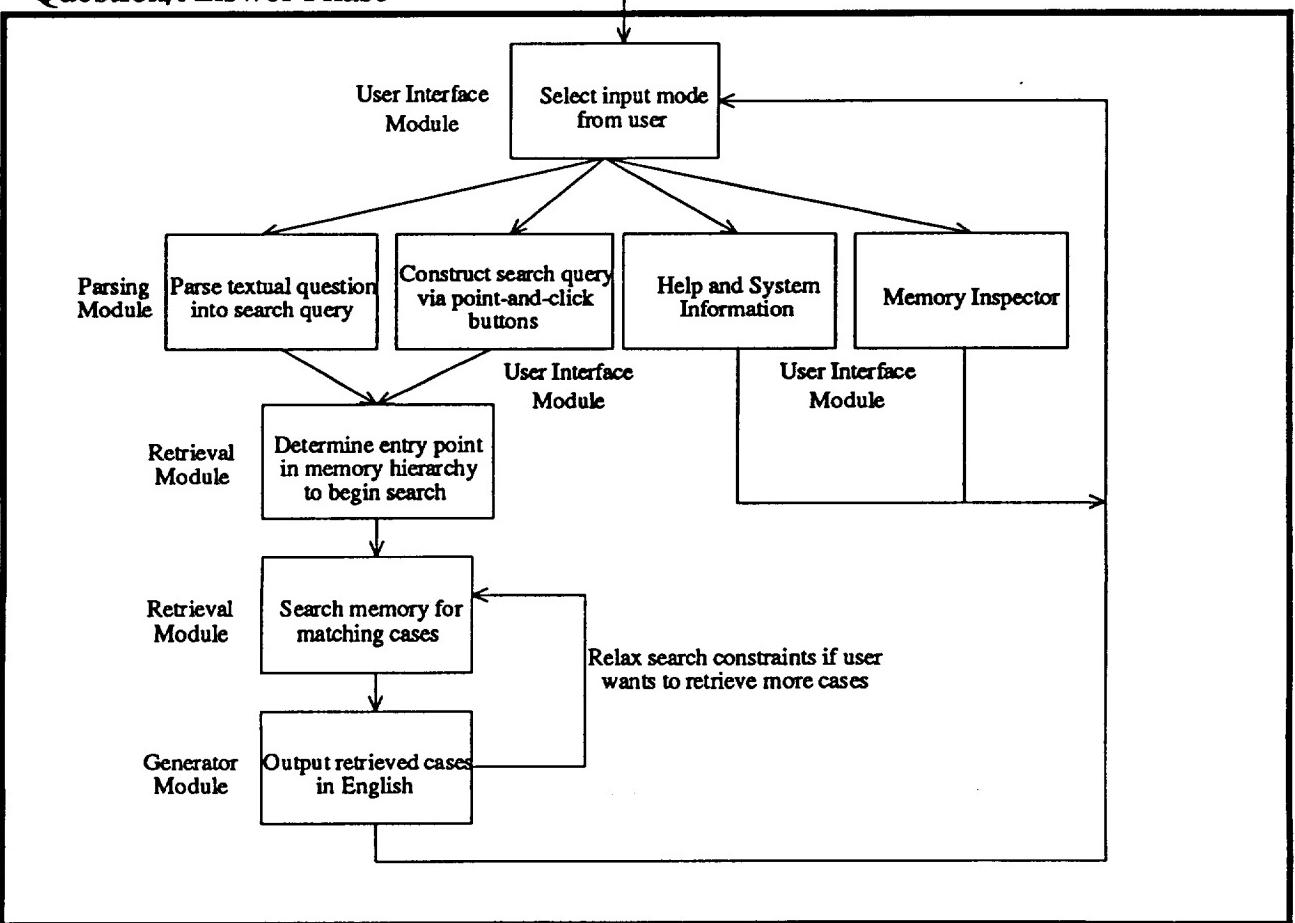


Figure 3.2
FANSYS Flow of Control

general MOPs occupy the topmost levels of the hierarchy. More specific concepts are indexed below their more general parents. This indexing scheme continues throughout the hierarchy, with the most specific instances of concepts ultimately indexed at the bottom. These specific concepts are called Instance MOPs; they are denoted I-MOP. All MOPs which are not instances (i.e., they are abstractions of some instance) are denoted M-MOP.

Although concepts in a MOP hierarchy are organized from the general to the specific, there are other relationships among MOPs used in FANSYS's memory. These relationships determine four memory hierarchies associated with the following types of links: ISA links, Slot-Filler links, Lexical links, and Generalization/Indexing links. For example, in figure 3.3 all four of these index types are depicted in relation to the event of powering up a DMS component. In this figure, the system's memory contains two instances of powering up an object. One of the instances involves a software entity called System-Management (SM) powering up a DMS component (I-M-Power-Up-Event.1), and the other instance involves the space-station crew powering up another DMS component (I-M-Power-Up-Event.2). These instances are connected to the abstraction of a general power-up-event (M-Power-Up-Event) via ISA links, which are depicted by arrowheads. These arrowheads do not indicate indexing or memory access direction, but serve only to differentiate the ISA links from other types of links. All links, including the ISA links, are bi-directional. ISA links connect sub-concepts to their parent concepts. In this example, a "System-Management powering up an object" is a type of "DMS-Entity powering up an object."

The Slot-Filler links connect concepts which are related to attributes of another concept. In figure 3.3, slots (i.e., attributes) of the Power-Up-Event include an actor performing the power-up action, an object being powered-up, preconditions for the event to occur, an action which is performed, and a state resulting from the action. All of these slots are constrained by their fillers (i.e., values). Only MOPs which are specializations of the filler may occupy that slot. In the Power-Up-Event example, only DMS-Entities may be the filler for the actor slot in a power-up event. Similarly, the Power-Off state is a precondition for the event, the Power-On state is a resulting postcondition, etc.

Lexical links connect all lexical information used to understand text with their respective concepts and index patterns. In the Power-Up-Event example, the word "power" is linked to one index pattern that contains "power" via lexical activation links. The figure shows two index patterns, one where an actor powers up an object, and the other where an actor powers down an object. The power-up pattern is linked to the power-up event through a lexical reference link. By traversing these links, the concept of power-up-event can be activated when the single word "power" is encountered while reading text. These indices are discussed in detail within the parsing section.

Finally, Generalization/Indexing (GI) links function similarly to both the ISA and slot-filler links, except they organize generalizations across cases and organize cases by their differences. GI indices are depicted in figure 3.3 by a triangle and dashed lines. In this example, the two power-up instances are indexed by the ways in which they differ. I-M-Power-Up-Event.1 is only accessible through the index of Actor=SM, while I-M-Power-Up-Event.2 is only accessible through the index of Actor=Crew. This simplified figure only shows one type of GI index; there may also be other indices for the object, precondition, action, and result slots. Furthermore, the

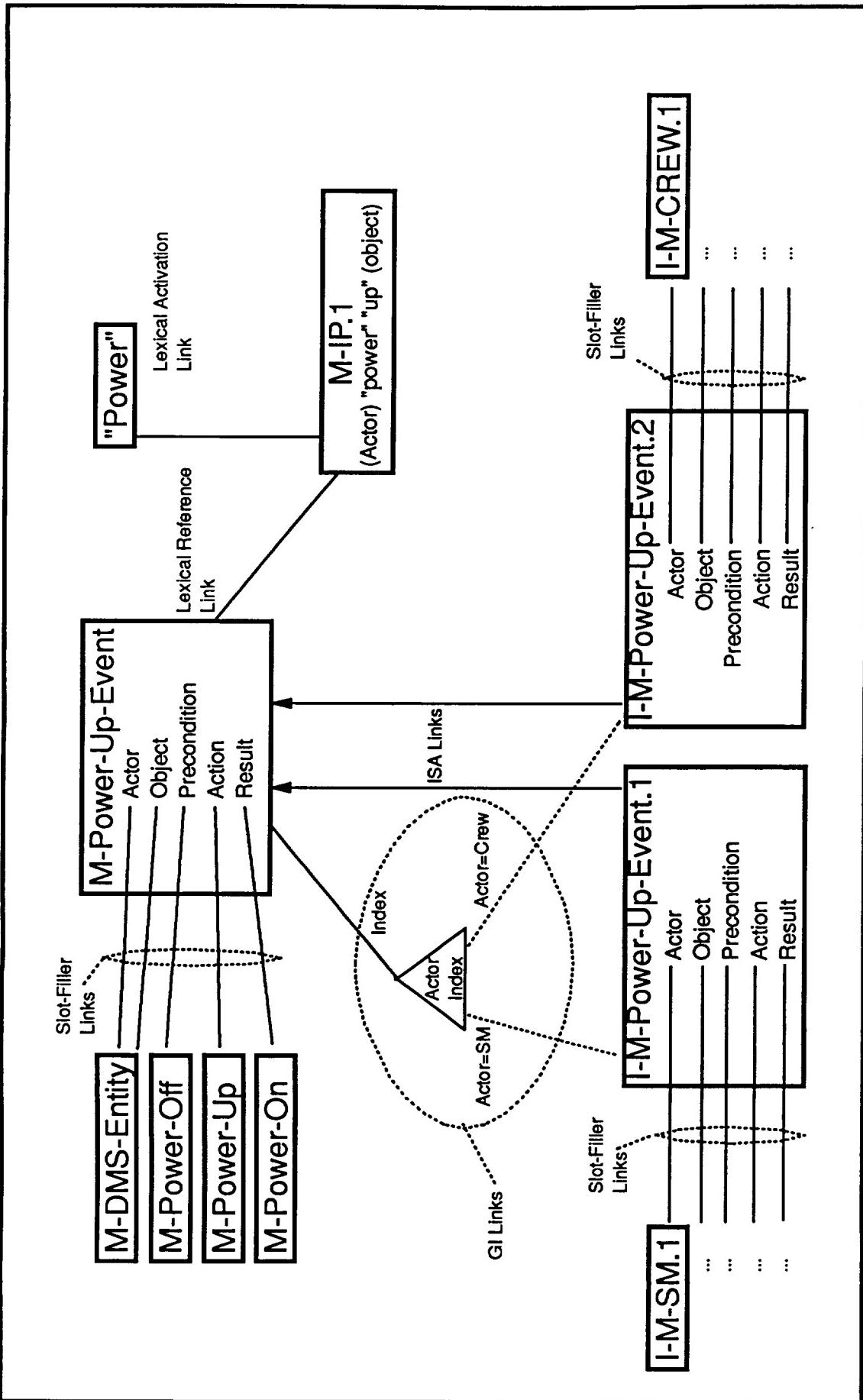


Figure 3.3
FANSYS Link Types for the Power-Up Event

GI indices are only depicted here by a triangle; however, the GI indices are not simple indices, but may be comprised of an entire hierarchy of MOPs. This GI hierarchy is very useful for quickly accessing memory during question/answering, and is discussed in detail within the memory retrieval section.

The previous example showed how the different links interact with one another to index and represent concepts in memory. However, figure 3.3 does not show how concepts are represented in memory as a whole. In contrast, figure 3.4 is a more comprehensive diagram which shows how the Power-Up-Event is represented in relation to M-Root, the conceptual abstraction of all concepts. In this figure, the dashed line encircles the example shown previously in figure 3.3. The Power-Up-Event is indexed under the more abstract category of Power-Events, which is a type of Event, which ultimately links to the Root. Similarly, a hierarchical organization is shown for index patterns, words, and the lexicon. There are also connections between ISA links and Slot-Filler links; i.e., an abstraction hierarchy also exists along the Slot-Filler links. These are depicted in the figure by the ISA links from I-M-SM.1, an instance which is the filler for a slot of I-M-Power-Up-Event.1. I-M-SM.1 generalizes up the ISA hierarchy to the abstraction of M-DMS-Entity, which is a filler for a slot of the more general Power-Up-Event. While not shown in the figure, similar links exist for the other fillers between abstractions and instances.

Figures 3.3 and 3.4 illustrate the different types of indices and how these indices organize concepts in memory. The same memory organization scheme is used for organizing entire failure cases, where each failure case contains the failure mode, correction procedure, detection procedure, failure causes, and failure effects for a given device. Failure cases are organized hierarchically through the ISA and GI links. These link hierarchies are shown for a portion of memory cases in figure 3.5. The GI index hierarchy is shown in dashed lines and triangles while the ISA hierarchy is shown with solid arrows; Lexical and Slot-Filler links are not shown. This figure shows how multiple instances of the System Management, I-M-SM.1 and I-M-SM.2, are indexed under the System Management MOP M-SM. An instance of the Power-On-Self-Test, I-M-POST.1, is indexed in an identical manner under the MOP M-POST. Similarly, a failure case regarding the Gateway, instance I-M-GW.1, is indexed under the Gateway MOP, M-GW. Entire cases are indexed in an identical manner under the Case MOP M-Case. Figure 3.5 also shows one other case, an instance of a ring concentrator failure case called I-M-RC.1.

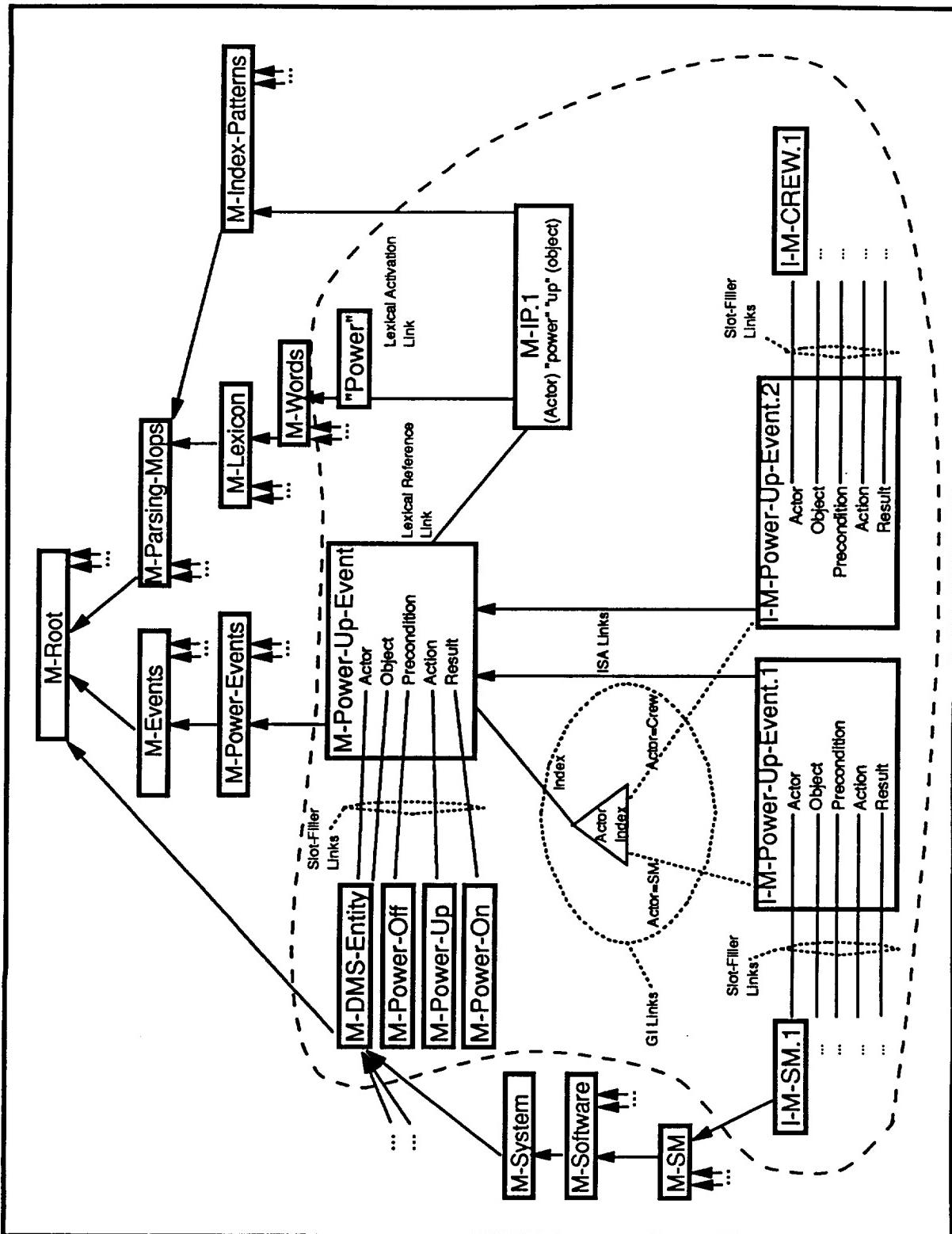


Figure 3.4

Sample Abstraction Hierarchy - Indexing of concepts within memory. The dashed oval is the Power-Up event shown in Figure 3.3. Ellipses denote other MOPs which may exist in the hierarchy.

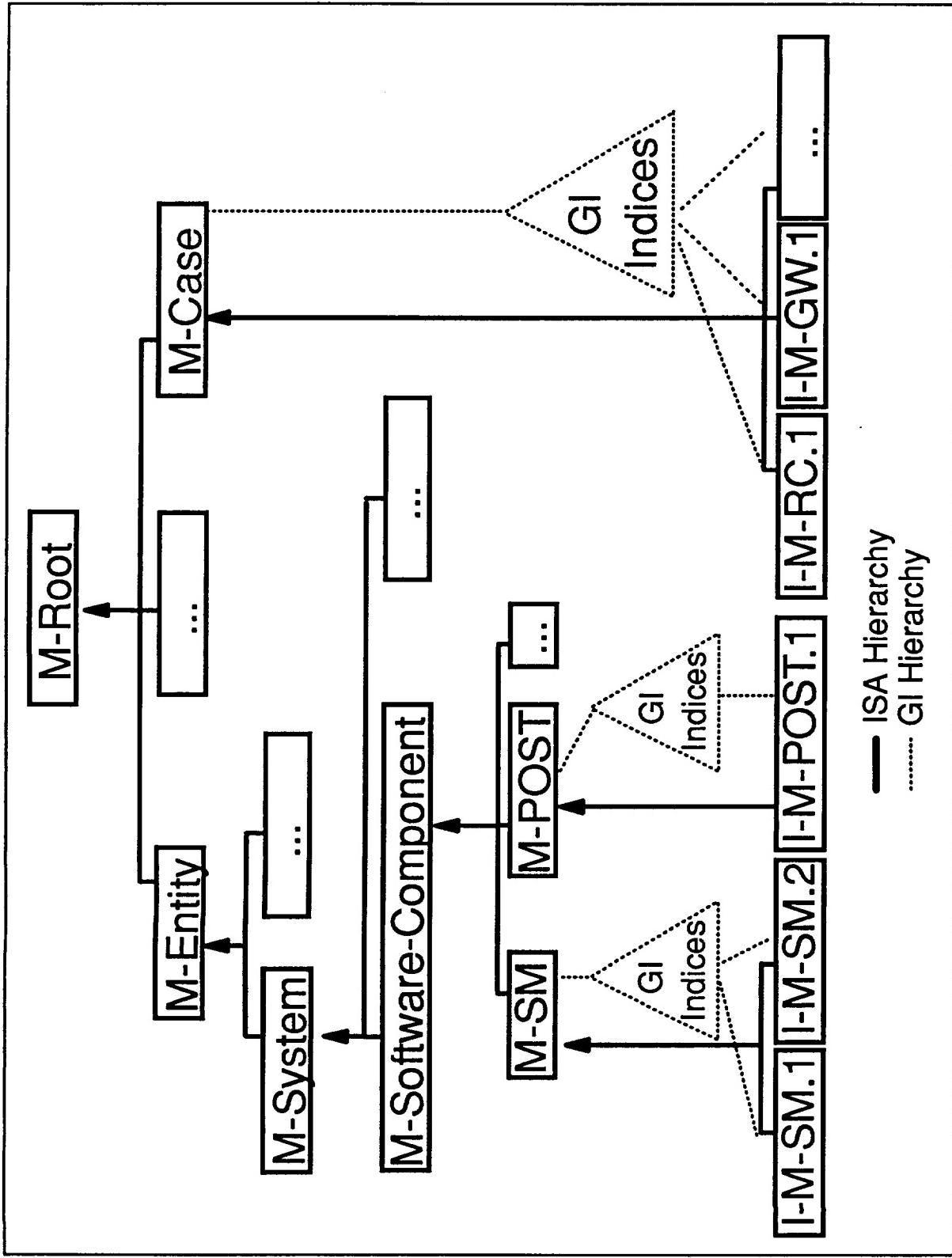


Figure 3.5

Partial memory abstraction hierarchy. The GI hierarchies are overlaid onto the ISA hierarchy to provide an alternate means to index and generalize among cases.

4. Domain Knowledge Representation

One of the first tasks confronting a researcher in constructing a natural language understanding system is to characterize the domain in which the understanding system is to operate (Schank, 1975, 1978). Once this domain has been characterized, a representational scheme must be adopted that will allow the researcher to encode that domain knowledge in a manner that permits the system to perform the required understanding task (Alvarado, 1990; Carbonell, 1981; Schank and Carbonell, 1979; Wilensky, 1986). In FANSYS, the model of domain knowledge used during text comprehension and question answering consists of two types of representational classes: one encompassing the specifics of the domain of space-station DMS faults, and one characterizing the fault cases which are described in the FMEA manuals (see figure 4.1). Although these classes differ in organizational function, they are each represented using the same frame-based representational structure of the MOP. The class that organizes DMS-fault knowledge characterizes the entities and relationships found within the DMS fault manuals, and enables FANSYS to parse case descriptions from such manuals, and perform limited inferences and reasoning within the domain of the space-station DMS. In contrast, the class that encodes the fault cases organizes a library of fault descriptions that is built up during parsing and queried during question answering. Both of these classes are indexed under M-ROOT, the top-most abstraction MOP which organizes memory. This section will consider these two classes of domain knowledge in detail.

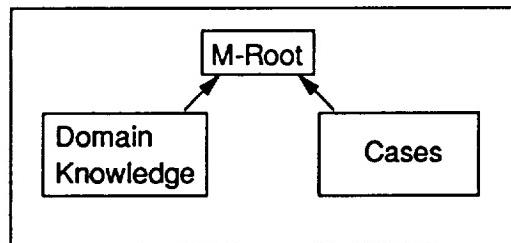


Figure 4.1
Knowledge Representation in FANSYS

4.1 The DMS Fault Domain

After having analyzed the domain of failure effects and procedures described in the FMEA manuals, it was determined that the DMS fault domain could be effectively modeled by five conceptual types of knowledge structures: entities, states, actions, events, and procedures (figure 4.2). These classes provide a detailed enough representational scheme of the domain to encode the events, relationships, and procedures underlying the textual descriptions of fault cases in the manuals. The case descriptions that are built by FANSYS are in turn founded upon these five types of domain knowledge. These types will be considered individually below.

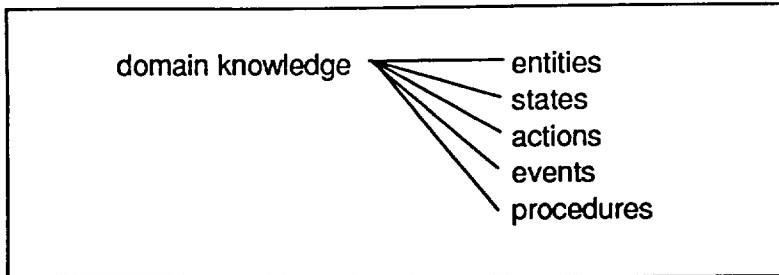


Figure 4.2
DMS Fault Domain Classes

4.1.1 DMS Entities

Entities are defined as the hardware and software components of the DMS system. There are two broad categories of DMS entities: systems and messages (figure 4.3). Systems are the physical devices of the DMS, such as an ORU (Orbital Replacement Unit), a token ring network, or an ORU's local system management software. Systems may be composed of hardware and software components, which may themselves be systems, to whatever granularity is required to model any of the textual descriptions.

As figure 4.3 shows, FANSYS recognizes several categories of systems. The mission components are the four systems that are effected by failures described within the FMEA manuals (for example, see the "Failure Effects" section, figure 2.1). The different networks of the DMS, as well as the DMS itself, are explicitly represented as two other categories of systems. The hardware components are the physical devices that comprise the DMS and its subcomponents, such as individual ORUs (e.g., ring concentrators and gateways) and the power systems. Finally, the software components of the DMS represent the software support systems, including system management software, error correcting procedures, and the like. These categories of systems effectively model the physical components of the DMS.

The second entity described in figure 4.3 is that of the message. Messages are the entities used in communication between or within systems. A token and a heartbeat message in a token ring network represent this type of entities. Messages are typically passed between systems within the DMS, carrying status or other information, and are differentiated from software systems in that they are not active agents within the DMS network. Rather, they represent passive constructs that are manipulated by systems.

Figure 4.4 shows a part of the ISA and Slot-Filler hierarchies for some sample entities. Notice that hardware components may be recursively defined as consisting of other hardware and software subcomponents. For example, a ring concentrator (a hardware component) may consist of a transmitter (a hardware component) and a local system management program, as well as other hardware and software subcomponents. Likewise, software components may consist of several other software subcomponents. (System management software typically has application management software as a subcomponent, for example, although this is not reflected in figure 4.4.) Such systems are defined to the level of granularity required to parse the case manuals; i.e. there is no need to represent a transmitter at the level of the atoms that compose it, since those atoms are not referenced within the FMEA manuals.

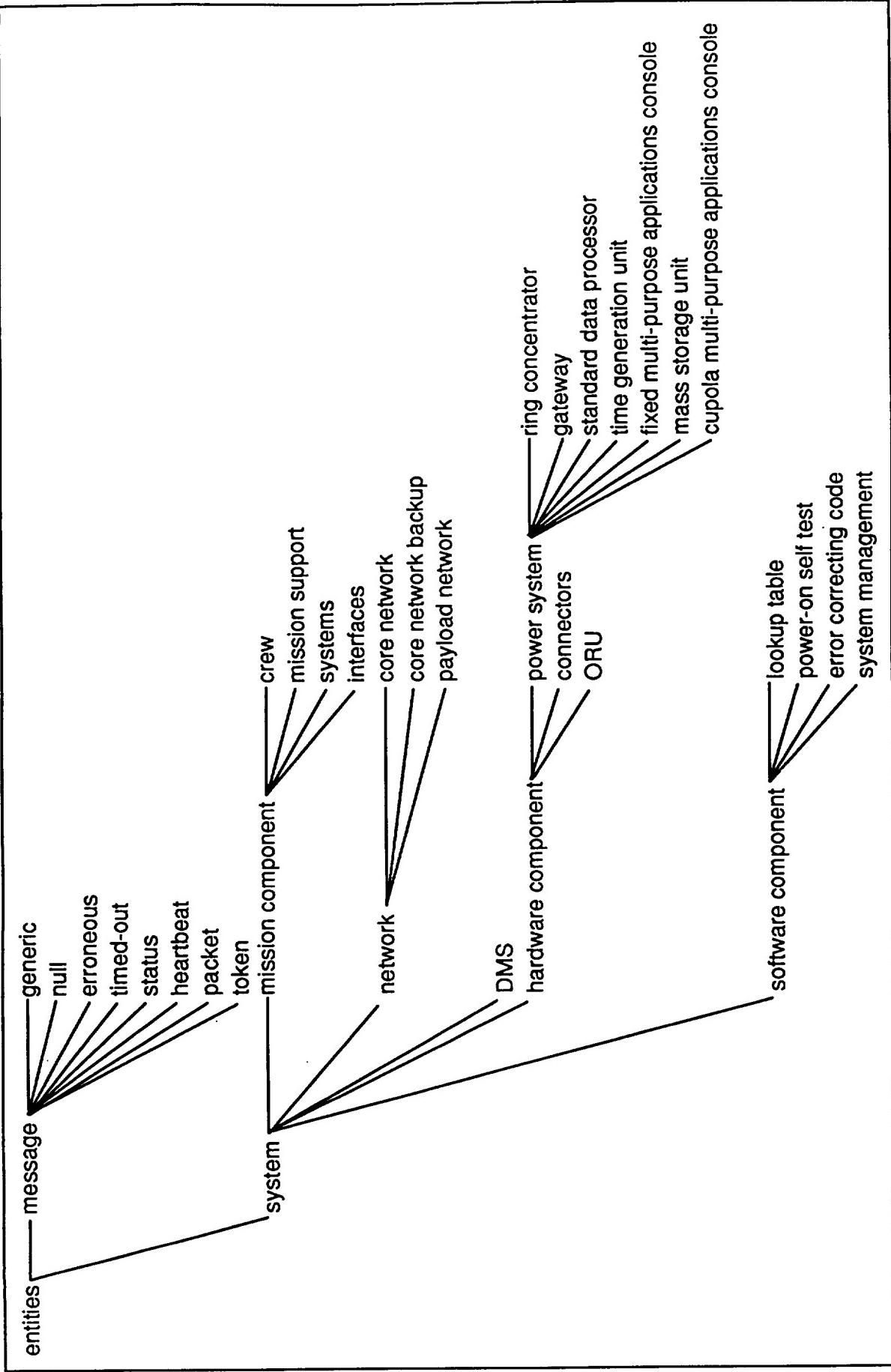


Figure 4.3
DMS Entities

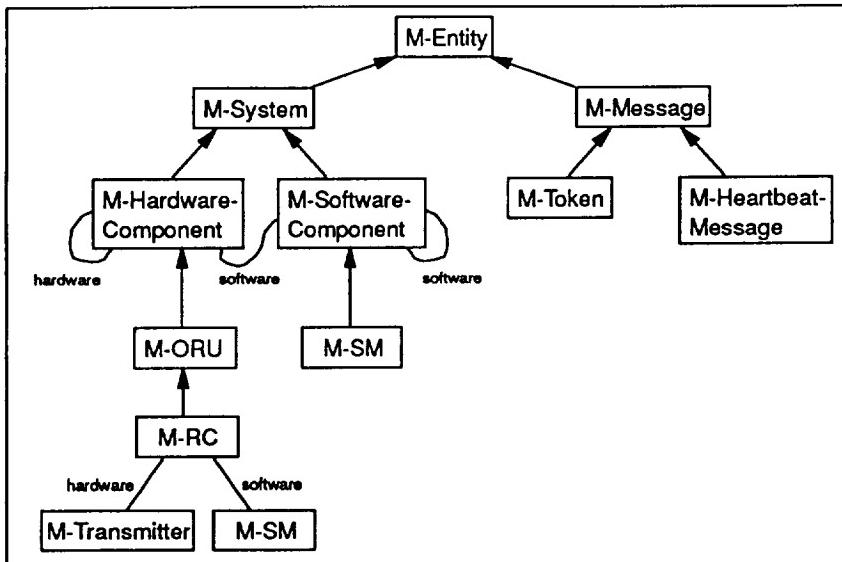


Figure 4.4
Entities: Memory Organization Example

To see how text is mapped into the class of entities, consider the sentence "system management reaches a time out limit for receipt of the network token from the ring concentrator." This text references two systems and one message: the ring concentrator and system management, and the network token, respectively. Figure 4.5 shows a concept instance for each of these entities from the text, and the actual text that maps into those instances. (The exact mechanism by which text is parsed into memory will be detailed later in section five on parsing.) Notice that as the ring concentrator is instantiated in memory, its subcomponents may be instantiated as well by inference. Thus, by explicitly representing concepts and their components in memory, the domain representation allows FANSYS to make inferences and connections not explicitly found within the text.

4.1.2 DMS States

A state represents a property of a DMS component. It may apply only to an individual component, or it may express some relationship between components. As shown in figure 4.6, there are eight states that are pertinent to the DMS domain. The first six encode physical properties for a specific DMS entity, including its structural integrity, whether it is operating within an acceptable temperature range, whether it has been contaminated, whether it is powered on or not, whether it is operable or faulty, and whether it has passed or failed a given diagnostic that has been performed on it. The other two states encode relationships between two entities of the DMS, such as a description of the other components to which a given device is logically or physically connected, and the current state of any communications between two entities. These states have been found to sufficiently characterize the state of the DMS system during the understanding process of the FMEA manuals.

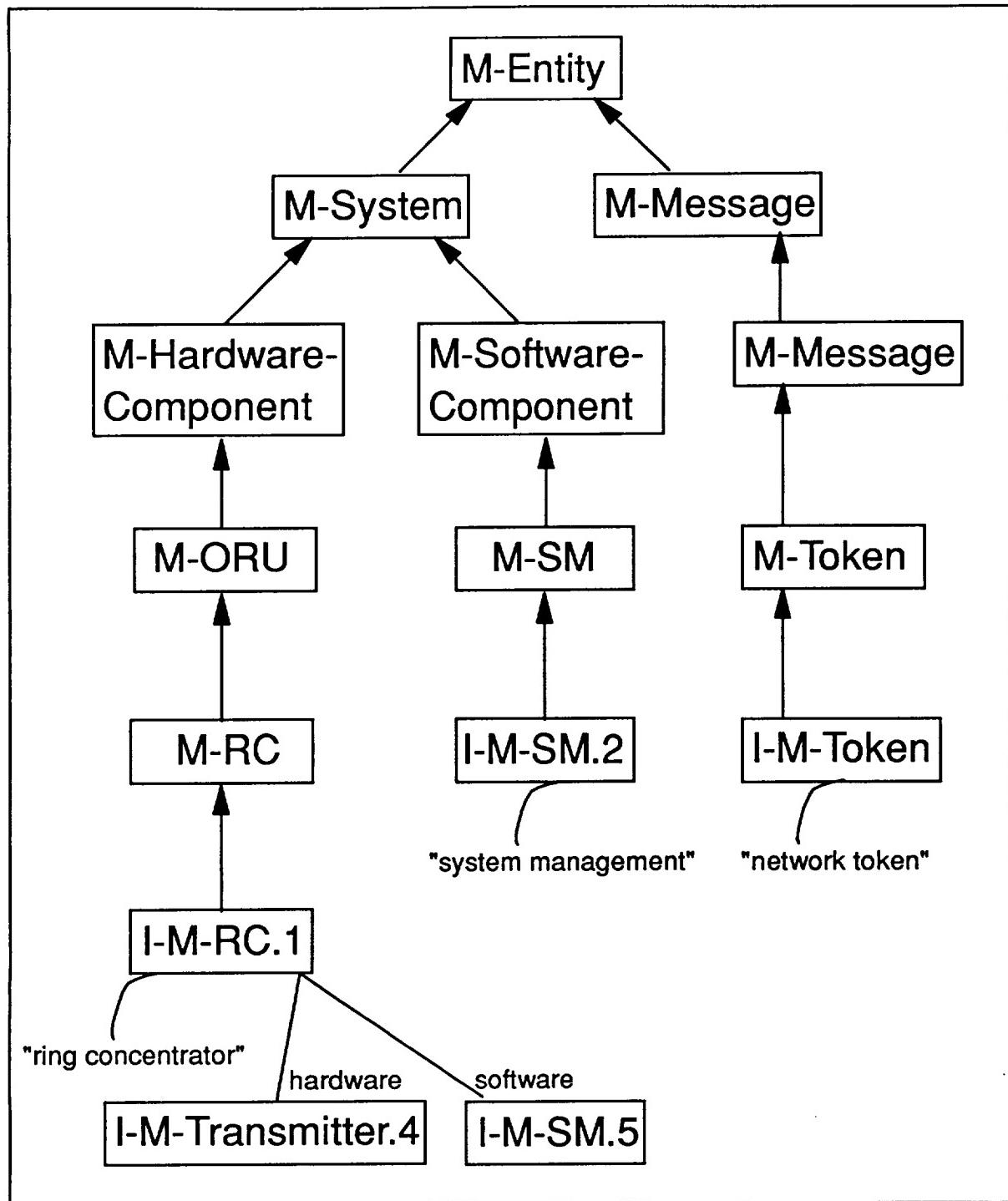


Figure 4.5
Entities: Textual Mapping Example

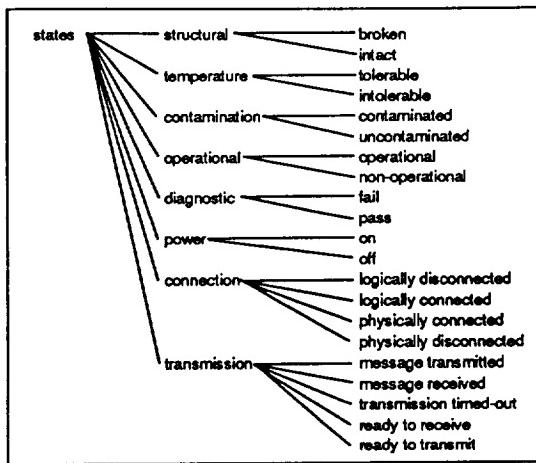


Figure 4.6
DMS States

Figure 4.7 illustrates a fragment of the ISA and Slot-Filler hierarchies for a few states. Notice that each of the states involving a single entity has a single slot detailing the object to which the state applies (e.g., m-power-on), while each of the states involving relationships between multiple entities has a couple of slots detailing the entities to which that state applies (e.g., m-physically-connected). These slots are expressed at an abstract level for each state, i.e., since slots are inherited down the ISA hierarchy, each of the subMOPs for that class will also inherit those slots.

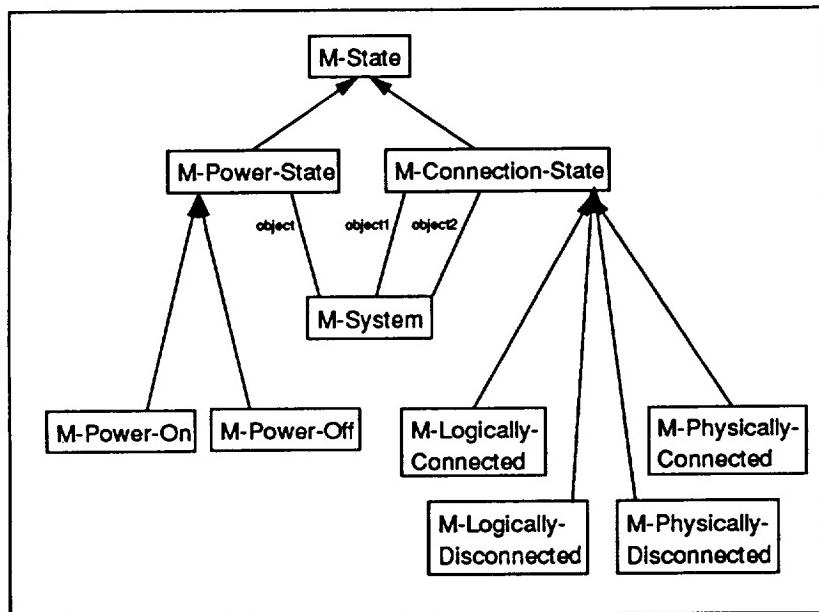


Figure 4.7
States: Memory Organization Example

The mapping of text into state structures is exemplified by figure 4.8, which shows the instances created for the parsing of two sentence fragments: "the power for the standard data processor is on" and "the ring concentrator is connected to the primary network." Notice that the

devices mentioned in the fragments map into entities, while the textual state assertions map into the states themselves. The text will generally specify the relationships of various subcomponents for a MOP to that MOP. For example, the first fragment specifies that the standard data processor fills the object slot of the m-power-on state.

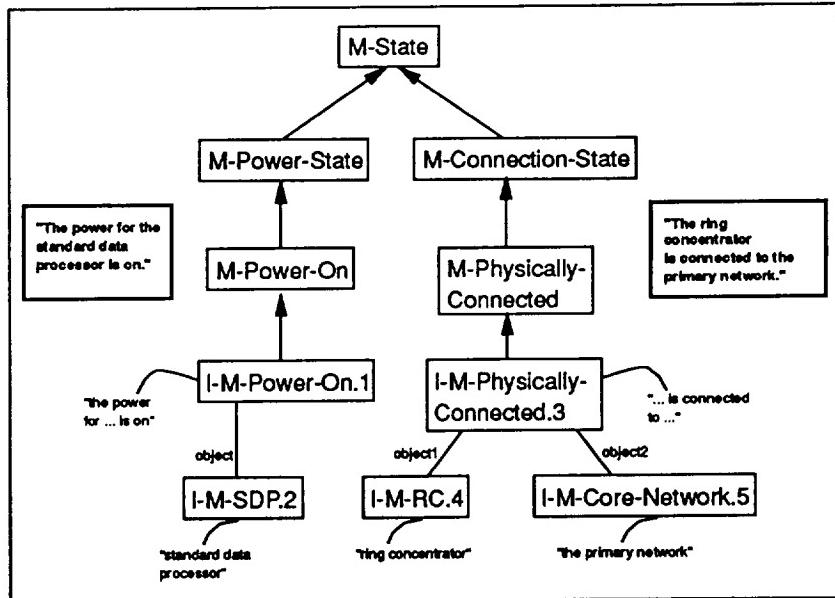


Figure 4.8
States: Textual Mapping Example

In practice, states are rarely mentioned explicitly in the manuals that were studied for the FANSYS project. Rather, they are usually left implicit as the results or preconditions of the events which are described within the manuals. Although left implicit in the text, these states must be explicitly represented in memory. For example, the result of powering on a ring concentrator is that the ring concentrator is in a state called m-power-on (perhaps with a precondition of being in the state of m-power-off). This relationship between states and events will be clarified shortly when the representation for events is discussed.

4.1.3 DMS Actions

Actions may be defined as acts performed by or on a DMS component. As with states, actions fall into a few well defined types, including powering actions, diagnostic actions, connection actions, and transmission actions (see figure 4.9). Powering actions involve turning on or off components, whereas diagnostic actions typically involve the performance of some sort of diagnostic test on a device. (For example, there is a diagnostic called the Power-On Self Test that is executed at powering-up time for some components, which determines if the component has become successfully operable.) Connection actions deal with establishing physical or logical connections between different devices. Transmission actions cover the transmission and reception of messages, and occurrences during communication such as the timing out of a device.

In addition to these types of actions, there are also some individual actions that describe operations on DMS systems. One of such actions represents the moving of something from one physical location to another, such as from the ground to the Space Station. Another such action

encodes the physical verification that a system is in the state that it is supposed to be in, e.g., that the connectors on a unit are properly connected. The last such action represents the looking-up of a backup unit in system tables by the system management for selection during the switchover to a redundant unit.

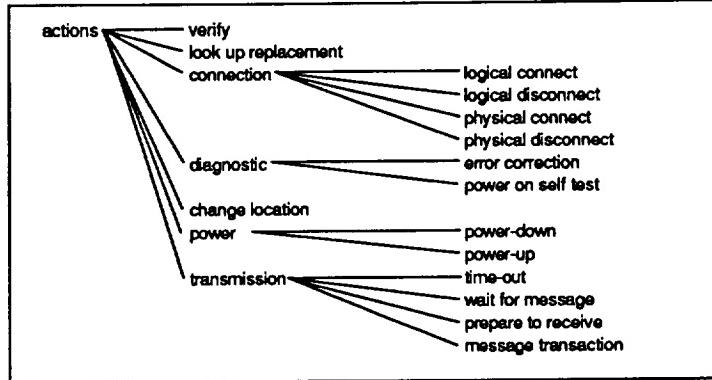


Figure 4.9
DMS Actions

Actions always involve an actor, and one or more objects that are acted upon. See figure 4.10 for some sample action MOPs, and the ISA and slot-filler hierarchies which characterize them. Note that while the powering actions have the usual object and actor slots that are inherited by all actions, the transmission actions have two additional slots detailing the sender and receiver of the transmission events.

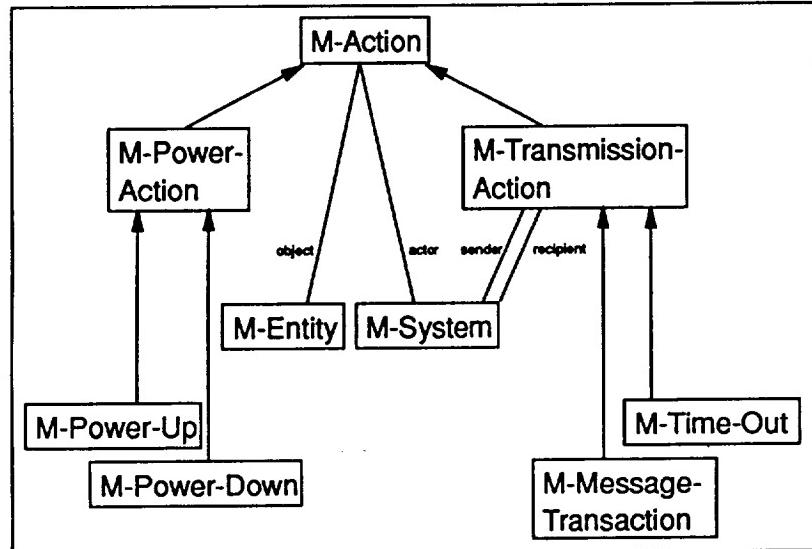


Figure 4.10
Actions: Memory Organization Example

In contrast to the prior types of domain representation that have been considered, actions are never mapped to directly by the parser. Rather, they are used as components of individual events; the events themselves are mapped to from the text. Actions thus represent atomic building blocks upon which events are founded. Figure 4.11 shows some sample action instances

that might be created during the recognition of the events which encompass them. Events will be considered in detail next.

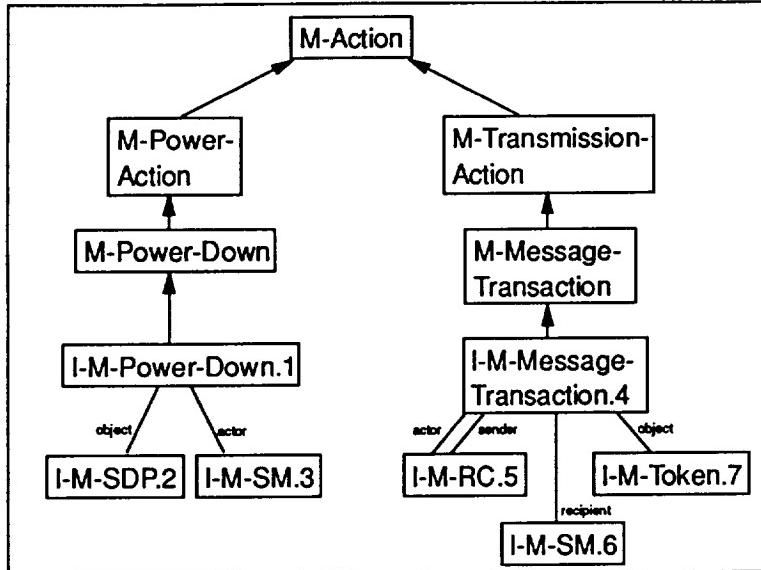


Figure 4.11
Actions: Instance Example

4.1.4 DMS Events

An event represents the occurrence of some action and the resulting state change that takes place because of that action. Each of the actions detailed in figure 4.9 has a corresponding event that describes the state transitions for that action; the categories of events are thus identical to the categories of actions (see figure 4.12). To illustrate this, consider the class consisting of powering events. Powering events involve transitions between power states based on the powering actions. For example, the event m-power-down-event packages the m-power-down action together with the precondition that the power be on initially (the state m-power-on) and the result that the power be off afterwards (the state m-power-off). The other classes of events likewise specify state transitions based on the actions to which they correspond. Since the precondition and results of events are seldom mentioned explicitly in a text, they must be explicitly represented in memory. This facilitates reasoning based on world knowledge and the generation of inferences regarding the current state of the DMS systems.

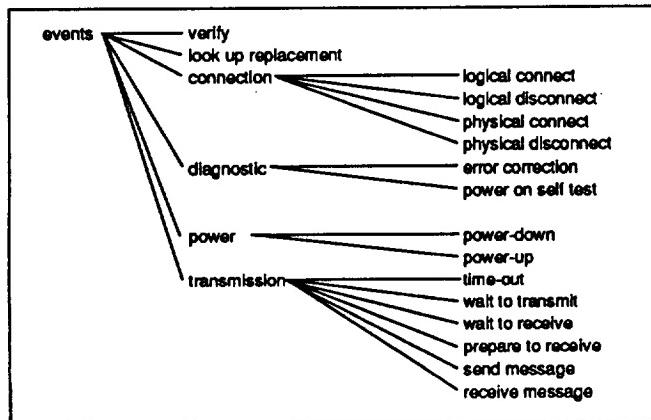


Figure 4.12
DMS Events

Since an event characterizes state transitions based on an action, an event may be defined as an action, that action's preconditions (which are each states), and the resulting states that follow from that action. Figure 4.13 depicts this slot-filler relationship, as well as other sample parts of the ISA and slot-filler hierarchies for the event class. Note that every event has one (or more) preconditions and results associated with an action at the abstract level. As the classes of event MOPs become more specified, the individual filler constraints also become more specified as to exactly which states and actions are involved in characterizing the associated MOP.

Text from the FMEA manuals that describe events map directly into the event MOPs, as demonstrated by figure 4.14. The sentence "the system management powers down the standard data processor" causes an instance of the m-power-down MOP to be created. The action, preconditions, and results associated with that MOP are inferred by virtue of the fact that the event was recognized to have occurred. FANSYS is thus able to track state changes during the parsing of text, a necessary step to the understanding task for which the system was designed.

4.1.5 DMS Procedures

The final and more complex conceptual type required to represent the DMS fault domain is that of the procedure. A procedure represents an ordered sequence of events or other procedures involved in some task described by the FMEA manuals. Figure 4.15 details the different categories of procedures (also sometimes called sequences-of-events within the FANSYS representation) defined within the system. Failure causes encode the events that lead to the failure of an ORU. These events are not actually specified within the manuals, save by named references like "piece-part failures" (see the "Failure Causes" section of the sample case manual in figure 2.1). For this reason, the actual set of steps for each of the failure causes is currently left undefined in the system. These steps could be fleshed out if more documentation that identified them became available.

The failure modes describe the events that occur when a failure is first detected. For example, a device may fail to become operable during the startup procedure for that device. Like the failure causes, the actual specifics of the procedure are left largely undefined, since they are not well specified within the FMEA manuals.

The failure detection procedures encode the specific steps used to detect the failure of a device within the DMS. For example, the m-next-node-detection procedure describes the case where a token circulating around a token ring network is not received by a specific ring concentrator; the implication is that the ring concentrator immediately upstream of it has failed. The last class of procedures is that of the failure corrections. These procedures describe the steps that need to be taken in order for a failure to be repaired. In the case of the failed ring concentrator just described, the correction procedure might be m-bypass-node, which consists of the automatic selectover to a redundant ring concentrator by the DMS system management.

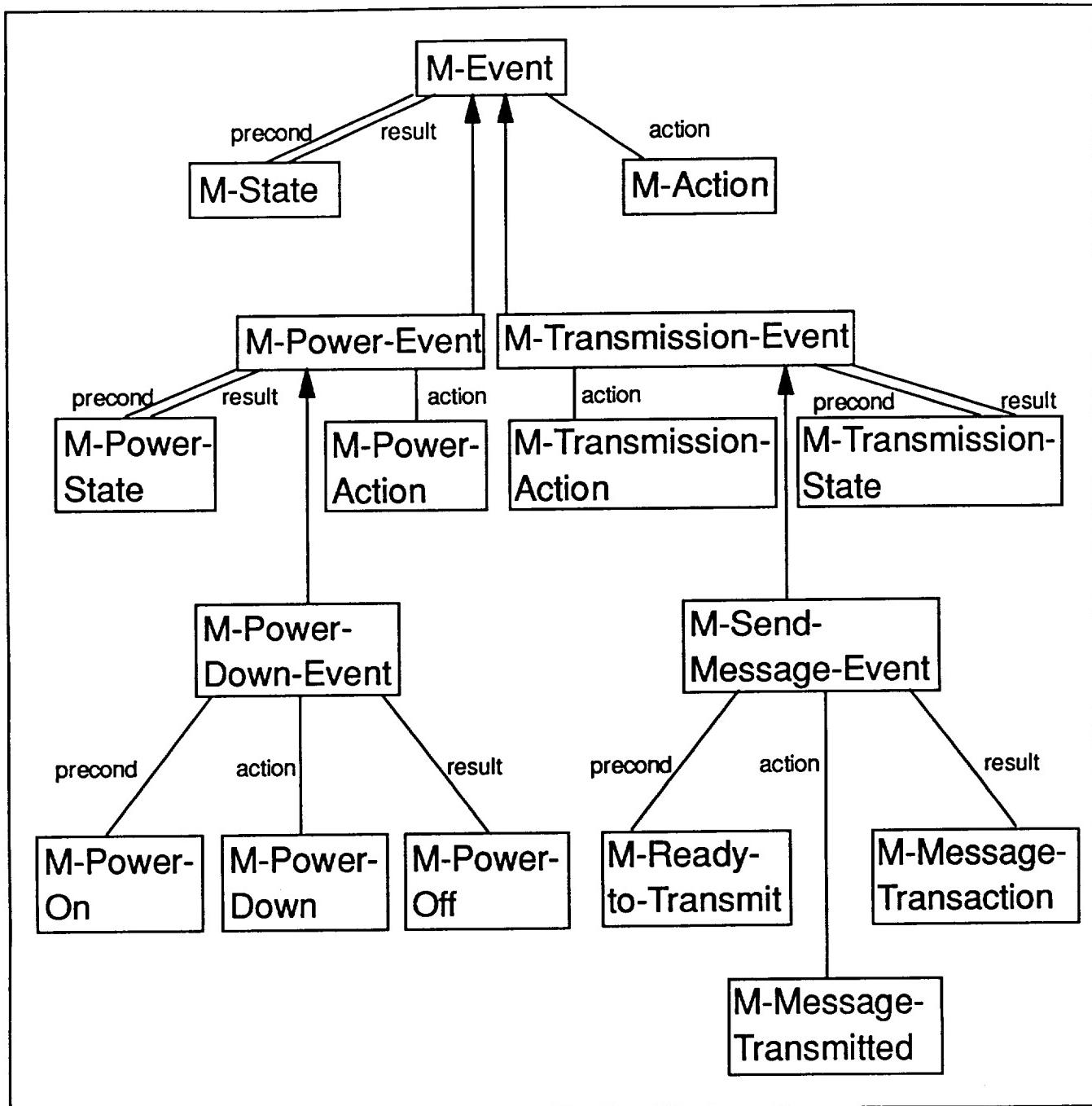


Figure 4.13
Events: Memory Organization Example

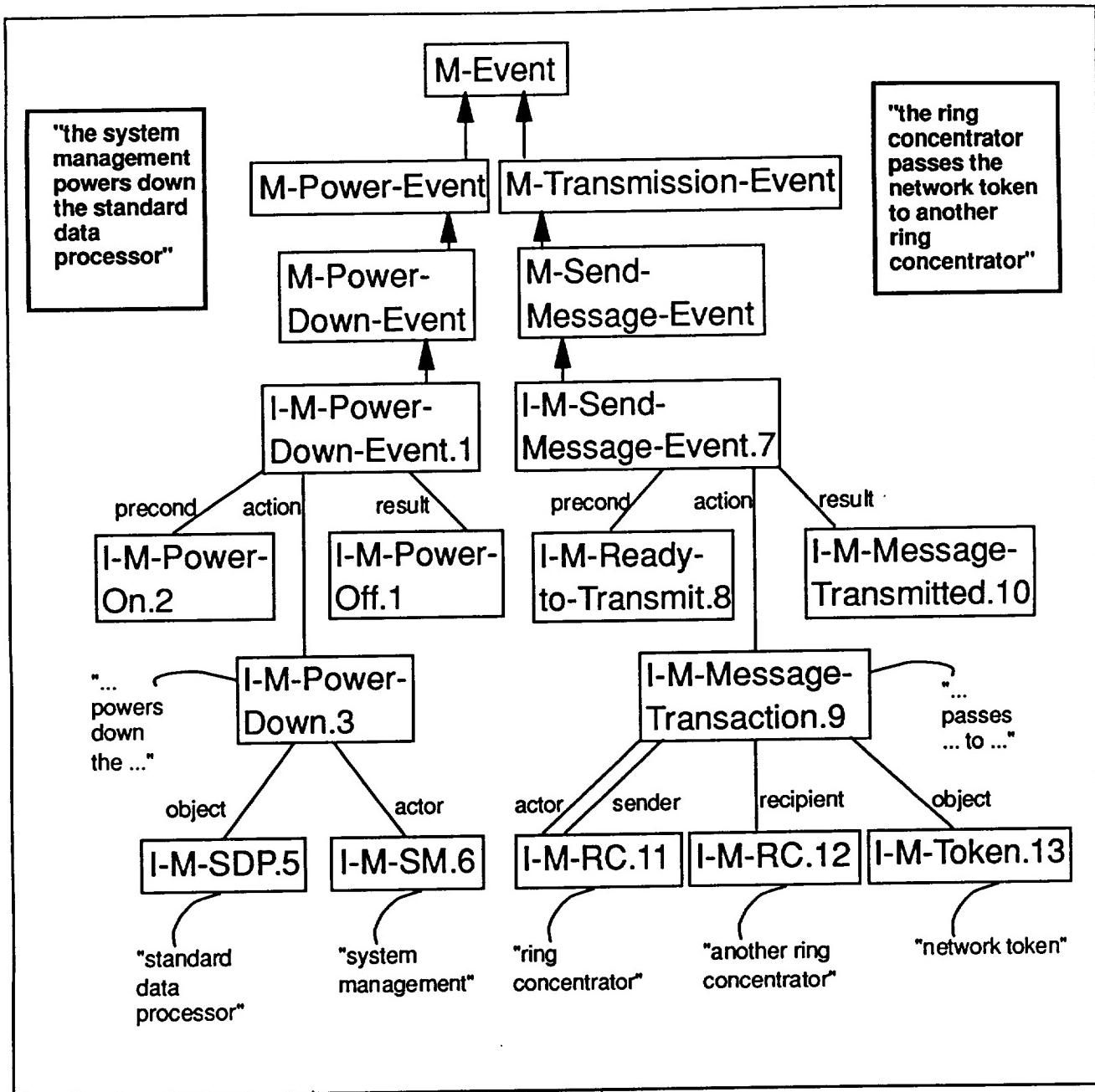


Figure 4.14
Events: Textual Mapping Example

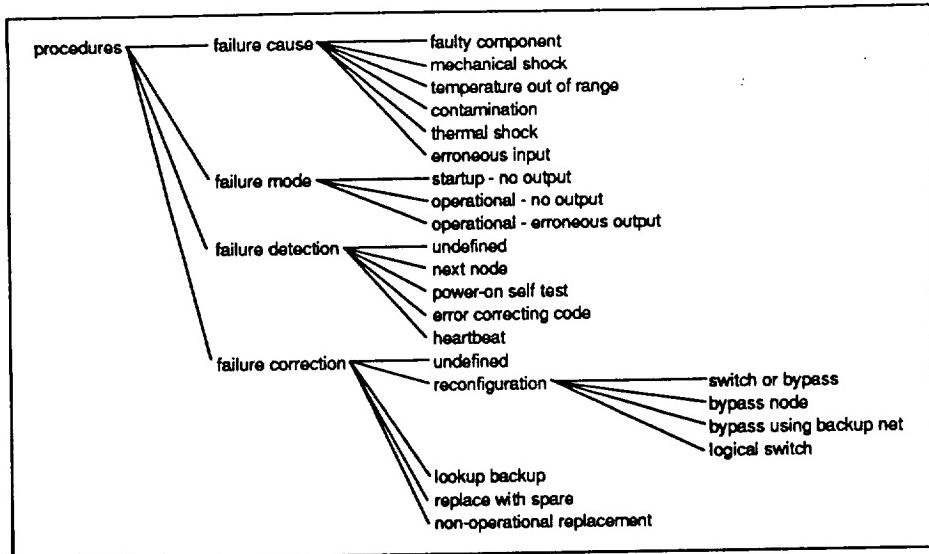


Figure 4.15
DMS Procedures

Figure 4.16 shows some simplified fragments of the ISA and slot-filler hierarchies for procedures. Like events, procedures have preconditions and results that are specified as state assertions; a procedure is in effect a complex series of state transitions (made by individual events) that lead from some starting set of states to some final set of states. For example, the preconditions for the m-next-node-detection procedure include the fact that the devices involved in the procedure are actually connected to the network physically and logically, while the result is that a failure has occurred. The sequence of steps details exactly what group of events (or other procedures) occurs in the procedure. Notice that a detection procedure actually consists of three subprocedures: the events in the detection process, the events necessary to verify that the failure has indeed occurred, and the final notification events to inform the DMS management of the failure. (Because only the detection steps are detailed in the manual cases with which the system currently deals, the notification and verification procedures are left undeveloped.)

One example of the detection process is illustrated in figure 4.16 by the steps for m-next-node-detection, which detail how a ring concentrator waits to receive a token from another ring concentrator, before eventually timing out. This process indicates a failure may have occurred in one of the ring concentrators. In addition to the slots specified in the figure, there are other slots not shown. For example, failure detection procedures have slots describing what component failed and which component actually detected that failure. These slots have been omitted from the figure due to spatial considerations; the interested reader is invited to examine the domain representation definitions included in Appendix C for complete details.

In order to illustrate how the representation of a detection procedure can be used to encode text, consider figure 4.17. The detection procedure for the first ring concentrator case consists of the following two sentences: "Indication of a RC failure is first detected by the next active node on the network. System management will reach a time out limit for receipt of the network token." As the figure shows, the first sentence is essentially a pointer to the class of detection procedure described by the text; in this case, it is a next-node detection procedure, in

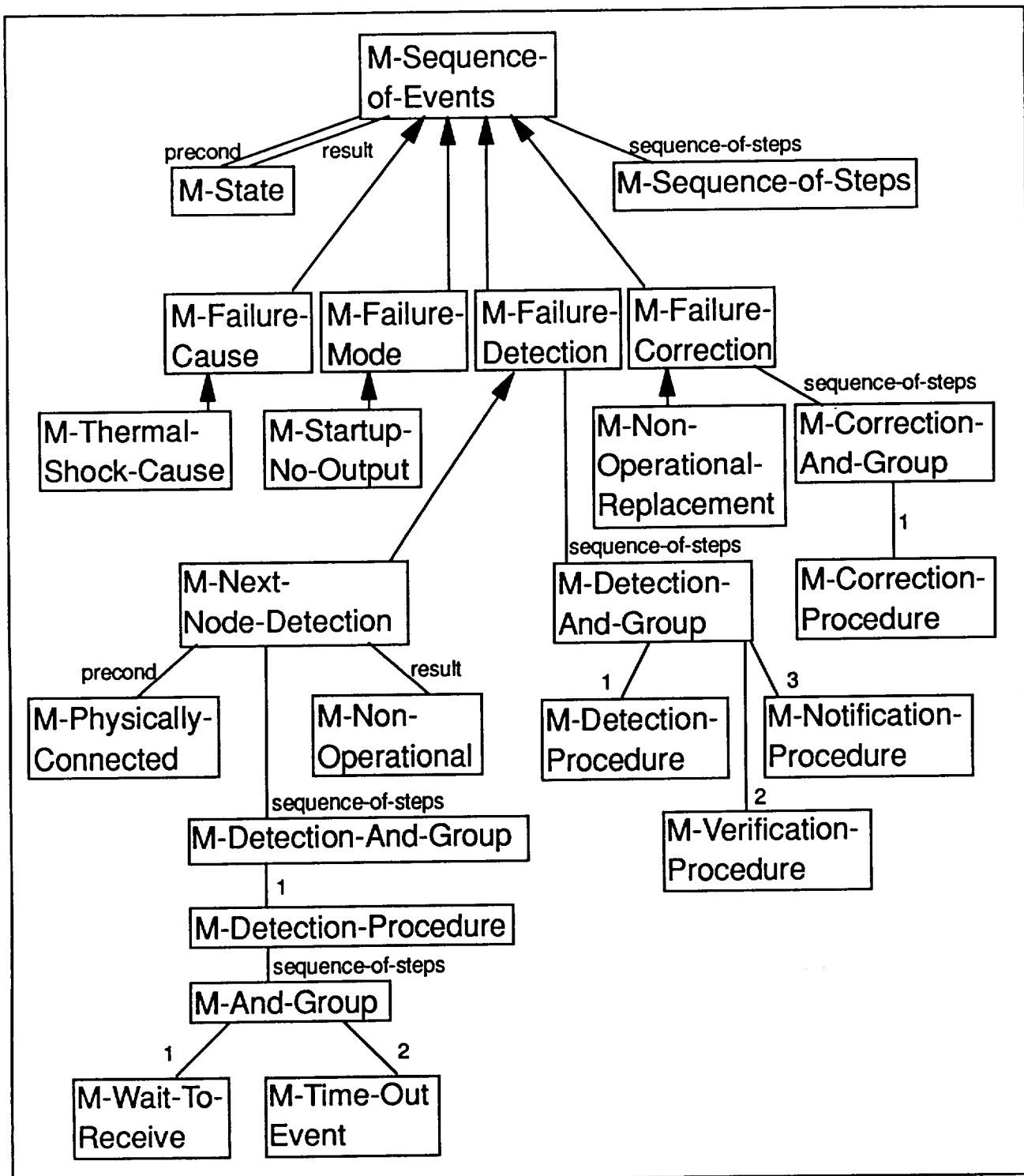


Figure 4.16
Procedures: Memory Organization Example

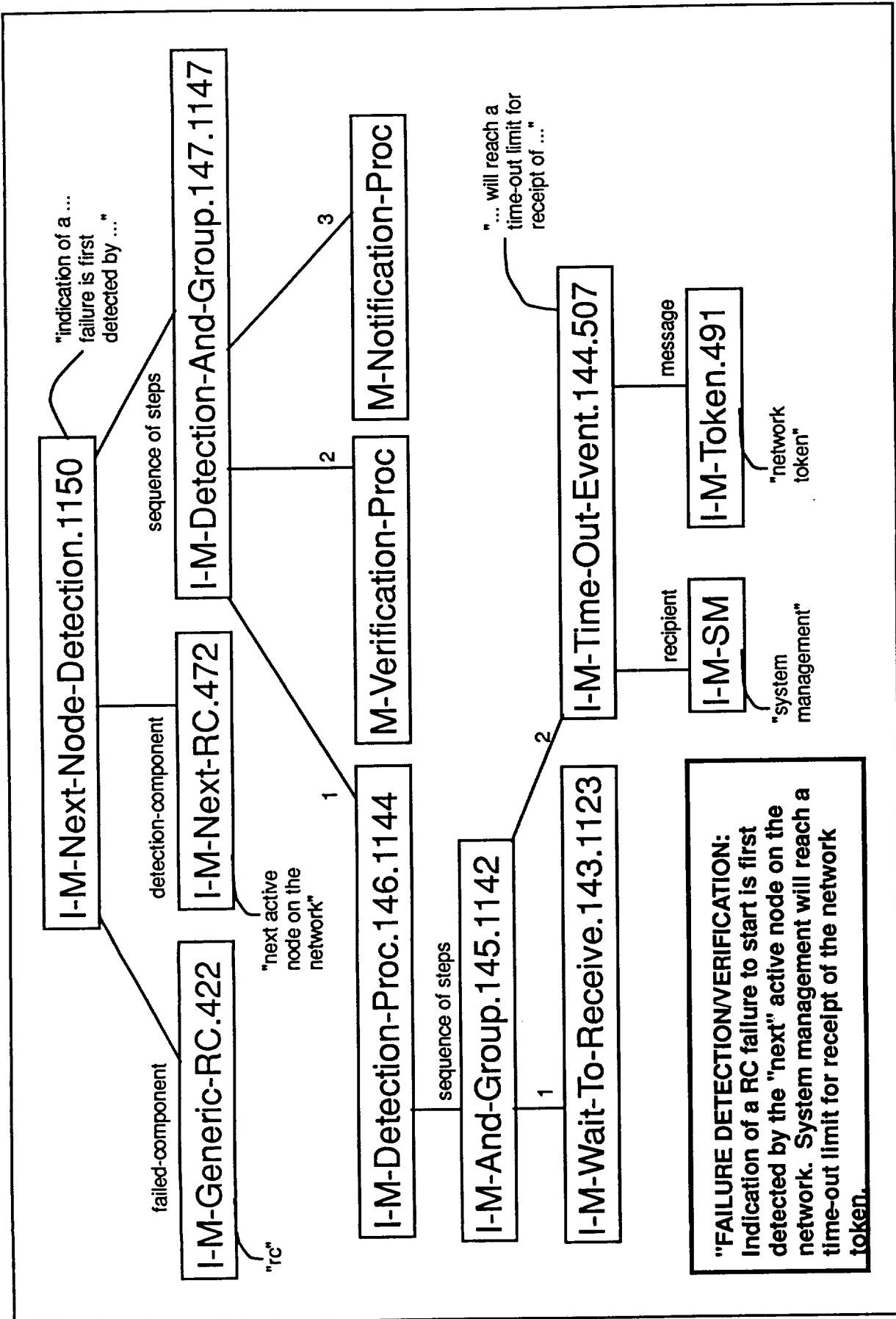


Figure 4.17
Procedures: Textual Mapping Example

which the node located downstream from a device in a network detects the failure of another device by timing out for the receipt of some message from that device. The second sentence references an actual step in the detection process: the timing out of the next node by failing to receive the token. Once an instance of the next node detection procedure has been instantiated, the rest of the representation for that procedure can be inferred, including the other steps in the detection process, the preconditions, the results, and so on (this is not shown in figure 4.17). The actual process of parsing text with procedural descriptions will be discussed in the section on Parsing.

4.2 Case Representation

Cases are the top-level organizing structures that package together entities, states, actions, events, and procedures into a representation of each failure case descriptions detailed within the FMEA manuals. A case description has slots corresponding to each of the main sections specified within a FMEA manual entry (see figure 2.1 for an example of such a text). Figure 4.18 shows the slot-filler hierarchy for a case description, including all of the slots that characterize a case (note that in practice there may be more than one of each type of slot in the case description). These include the failed component (an ORU entity), the failure causes and mode (procedures), the detection and correction procedures, and the failure effects (represented as a type of correction procedure). This case representation was determined by examining and characterizing the content of the FMEA manuals that the system is designed to parse.

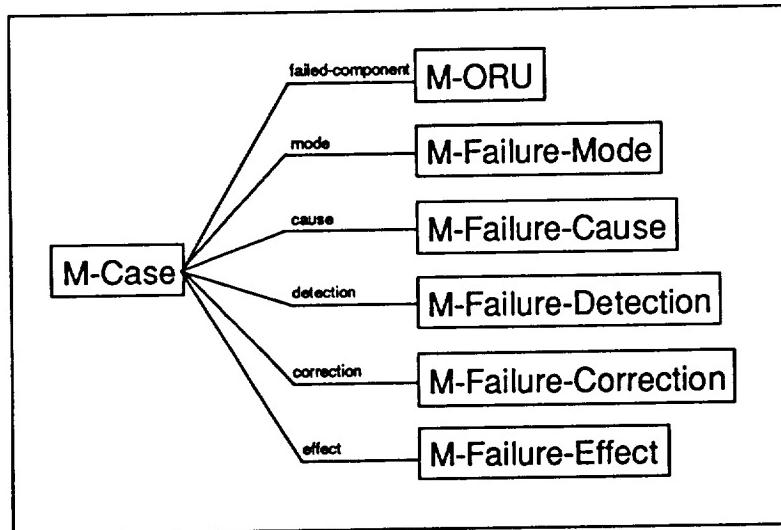


Figure 4.18
Representing a Case in FANSYS

Consider the text describing the first failure case for a ring concentrator given in figure 2.1. The next series of figures shows how each of the pieces of that manual text map into the memory structures that organize them. Figure 4.19 shows the mapping of the failed item name. The "item name:" tag is the stereotypical method in the FMEA manuals of specifying that what follows will be the name of the item that failed. When parsed, that tag provides a context which generates expectations to the system about what will occur next in the text. This is similarly the case for other tags like "failure mode:" and "failure effect on:". This process will be detailed in the section on Parsing.

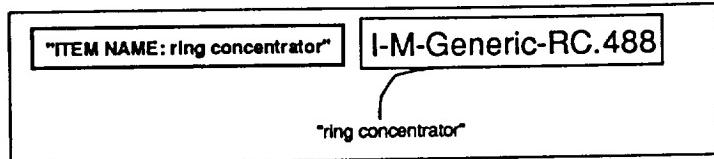


Figure 4.19
Failed Component Mapping for RC.1

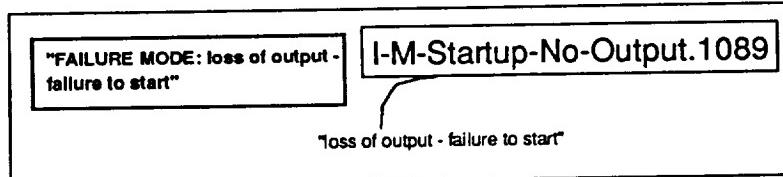


Figure 4.20
Failure Mode Mapping for RC.1

Figures 4.20 and 4.21 demonstrate the mappings of the failure modes and causes. These mappings are also fairly stereotypical in format; the text maps straight into the procedures which underlie it. However, the representation contains more information than the text for each of the modes and causes, and such an information is inferred when the individual MOPs are instantiated at parse time.

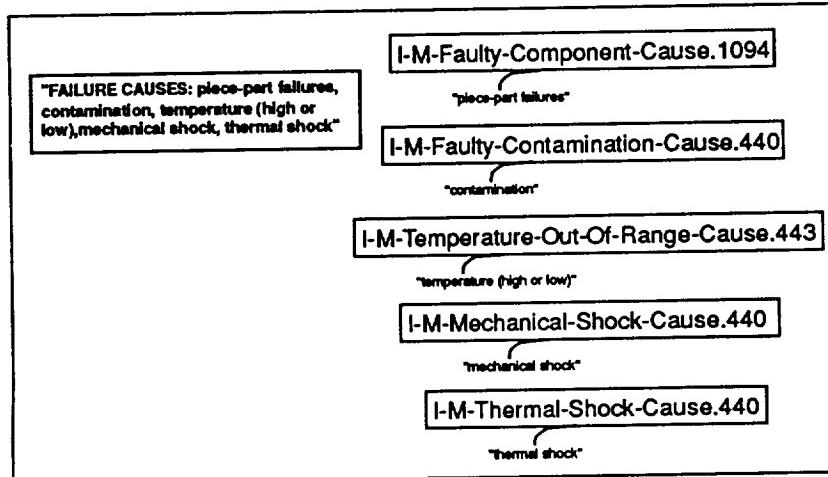


Figure 4.21
Failure Cause Mapping for RC.1

Figure 4.22 is a recapitulation of the figure used in the domain representation section on procedures, demonstrating the mapping of the detection procedure m-next-node-detection. As mentioned previously, the first sentence acts as a pointer to the type of detection procedure being described, while the second sentence provides information for instantiating the specific procedure.

Figure 4.23 is an example of the representation of the textual description of a procedure that has no steps explicitly specified; instead, the text just talks about the procedure in abstract terms, leaving the system to infer the actual steps. In this figure, the first sentence maps into a

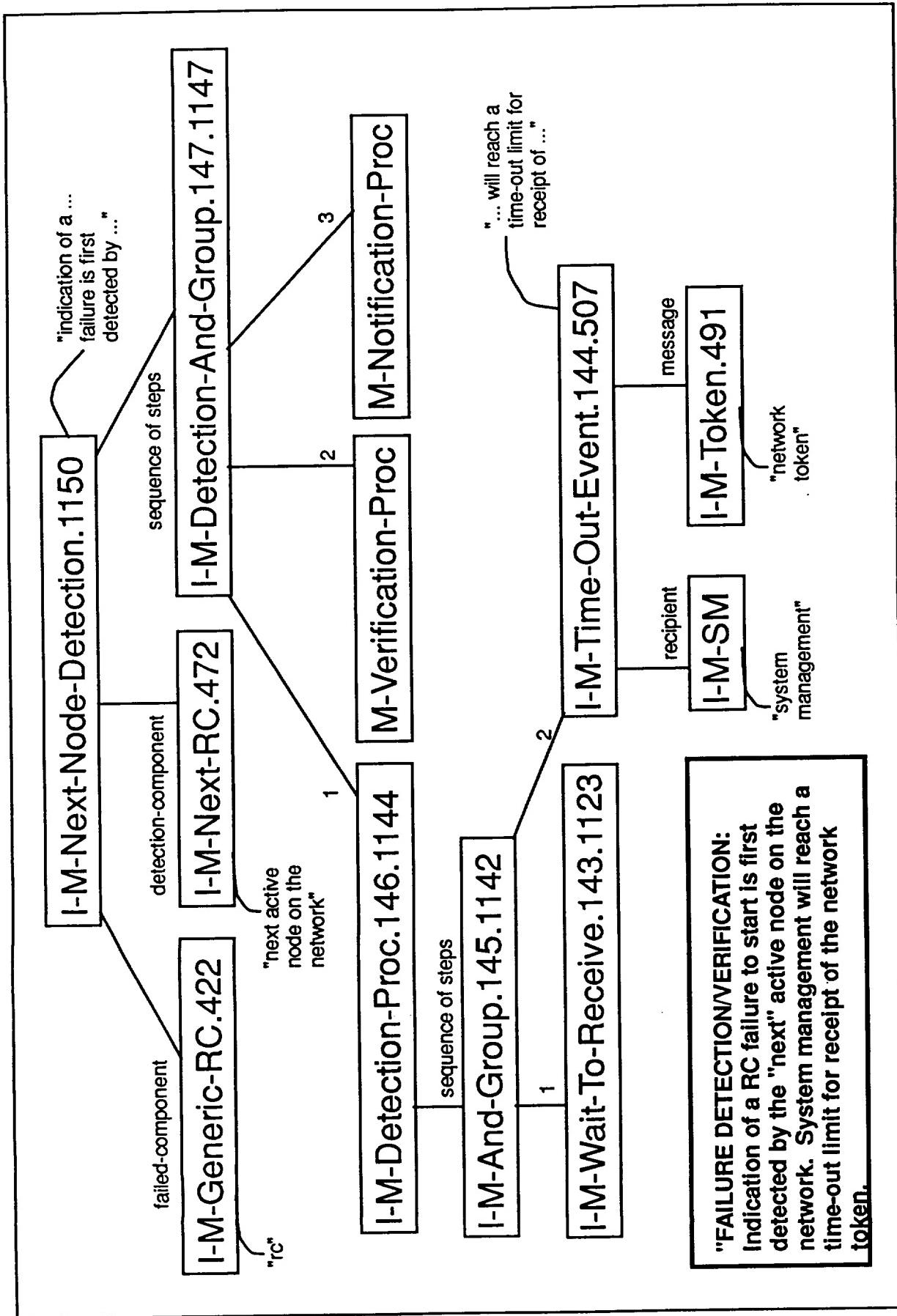


Figure 4.22
 Failure Detection Mapping for RC.1

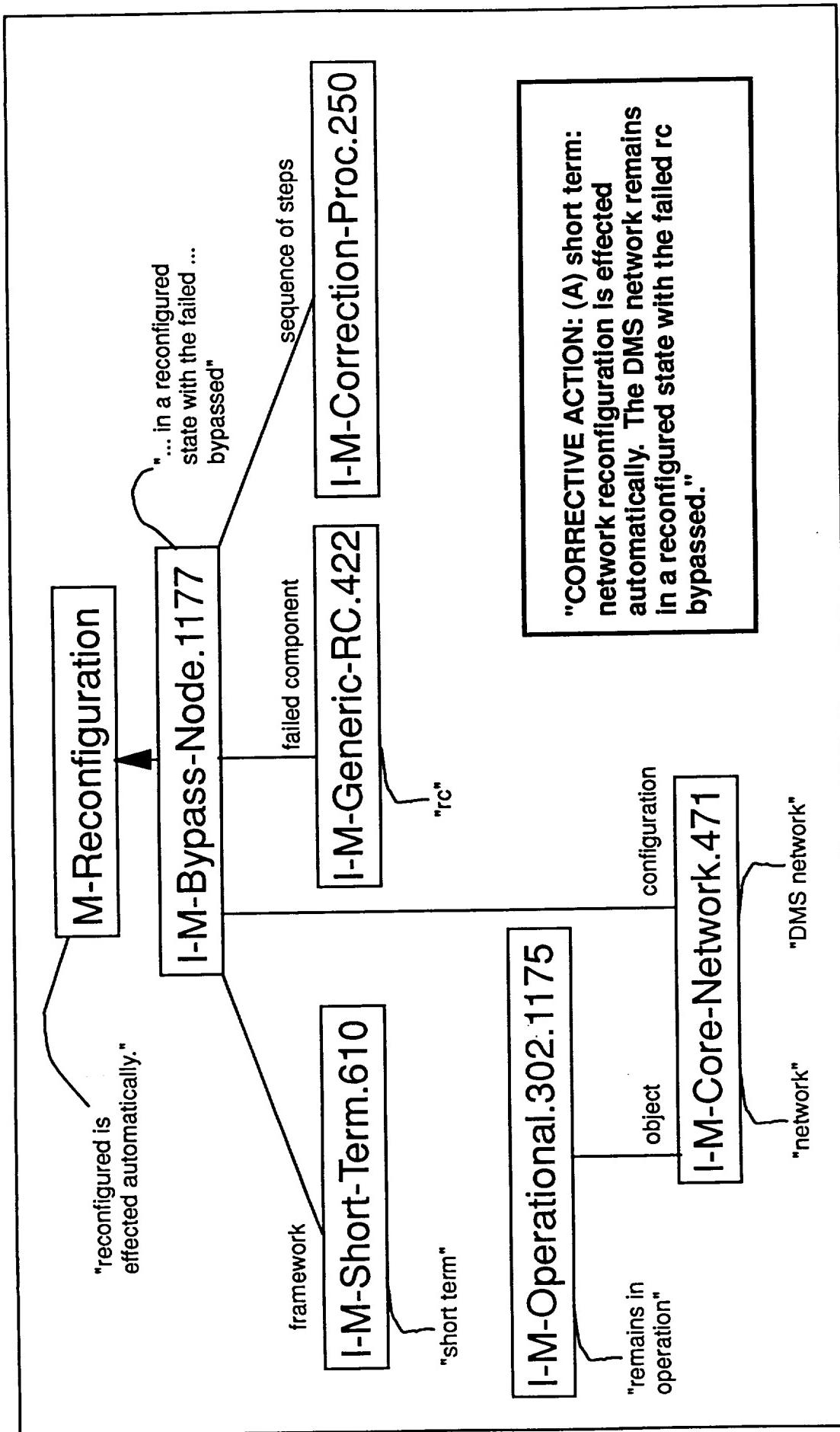


Figure 4.23
 Failure Correction Mapping for RC.1 (Short Term)

very broad category of correction procedures involving the reconfiguration of a network. The second sentence specifies which procedure in that class is being referenced.

Figure 4.24 provides an example of a procedure wherein only the steps of the procedure are referenced, since both sentences describe individual steps of the m-non-operational-replacement correction procedure. As the last three figures suggest, the parser must be able to construct and recognize a procedure when any combination of pointers and individual steps occur within the text.

Figure 4.25 shows the mapping of the failure effects. In each of the manual entries examined, there was generally no effect on the mission components (this mapped to an instance of the null-procedure). In some cases, a justification is given that the correction procedure alleviated any possible effects (the first effect of figure 4.25, for example). In such cases, the text is mapped to the correction procedure which is used as that justification.

Figure 4.26 ties all of the preceding figures together into a single case representation. This is the complete characterization of the first ring concentrator case that is built by the system during the parsing of that case. The other cases have similarly been represented by hand-coding them using the same representational scheme (see Appendix A for details), although the parsing process should be largely the same as the example that is handled by FANSYS. The actual logistics of the parsing process are described in the next section.

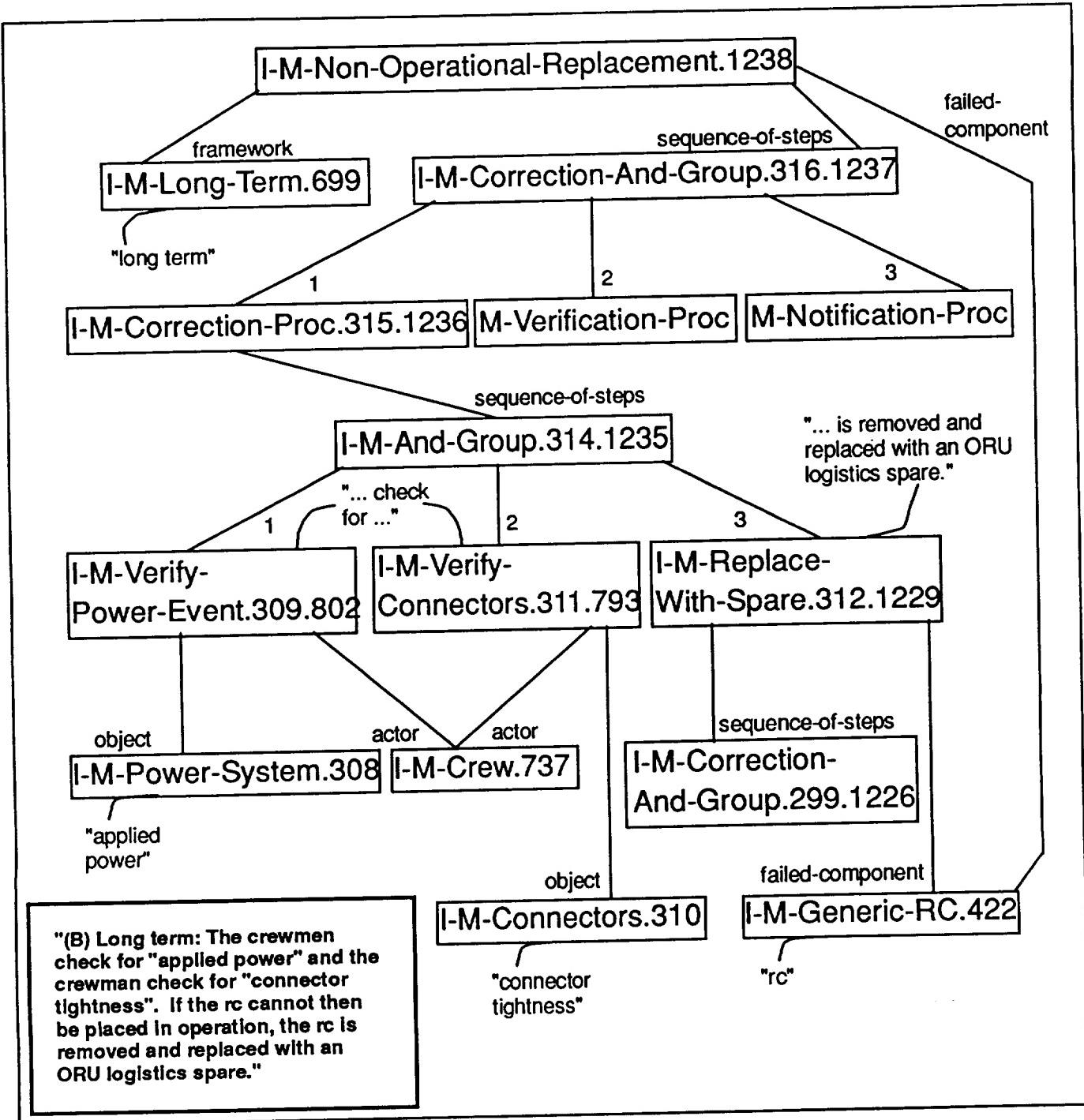


Figure 4.24
Failure Correction Mapping for RC.1 (Long Term)

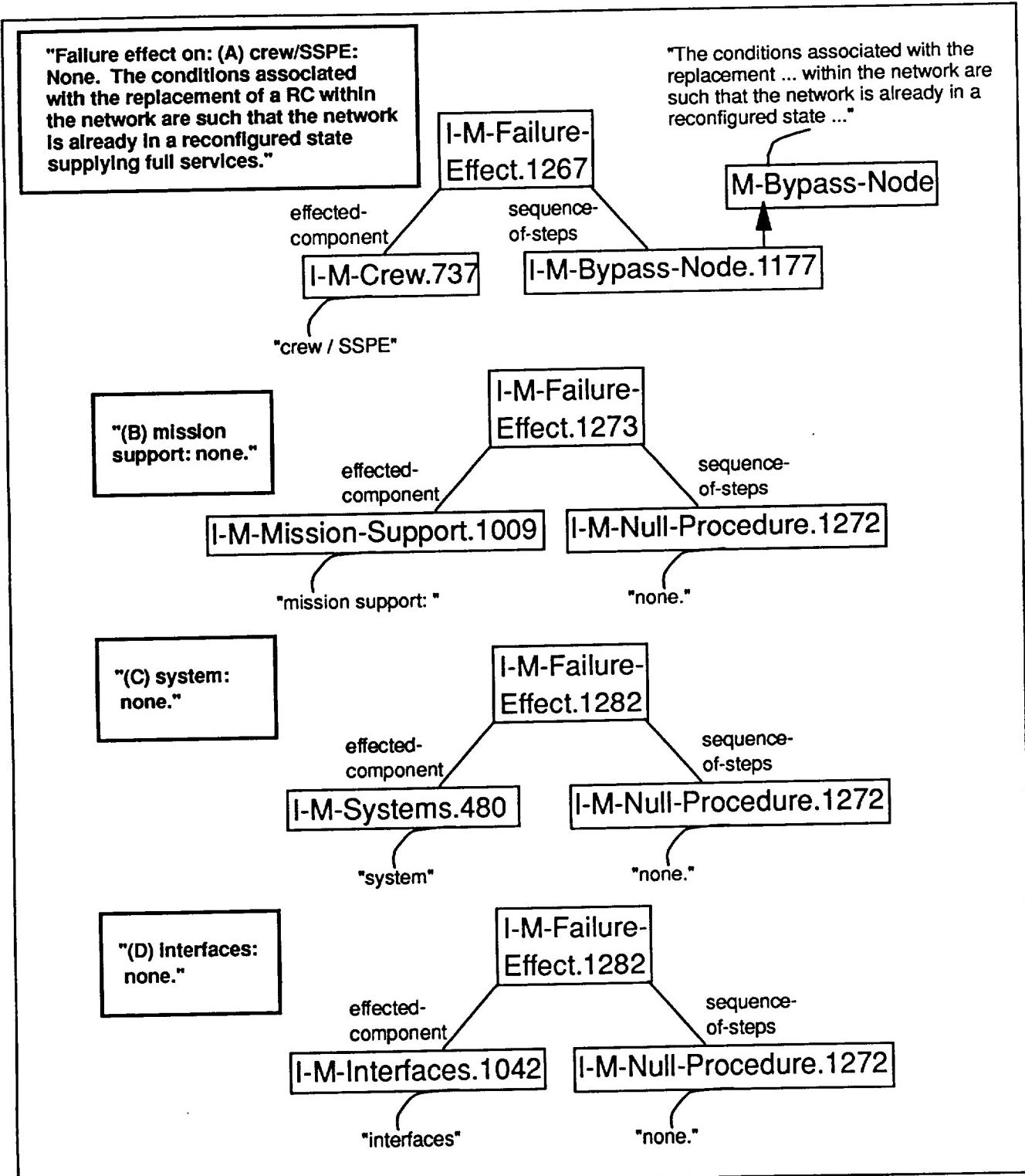


Figure 4.25
Failure Effects Mapping for RC.1

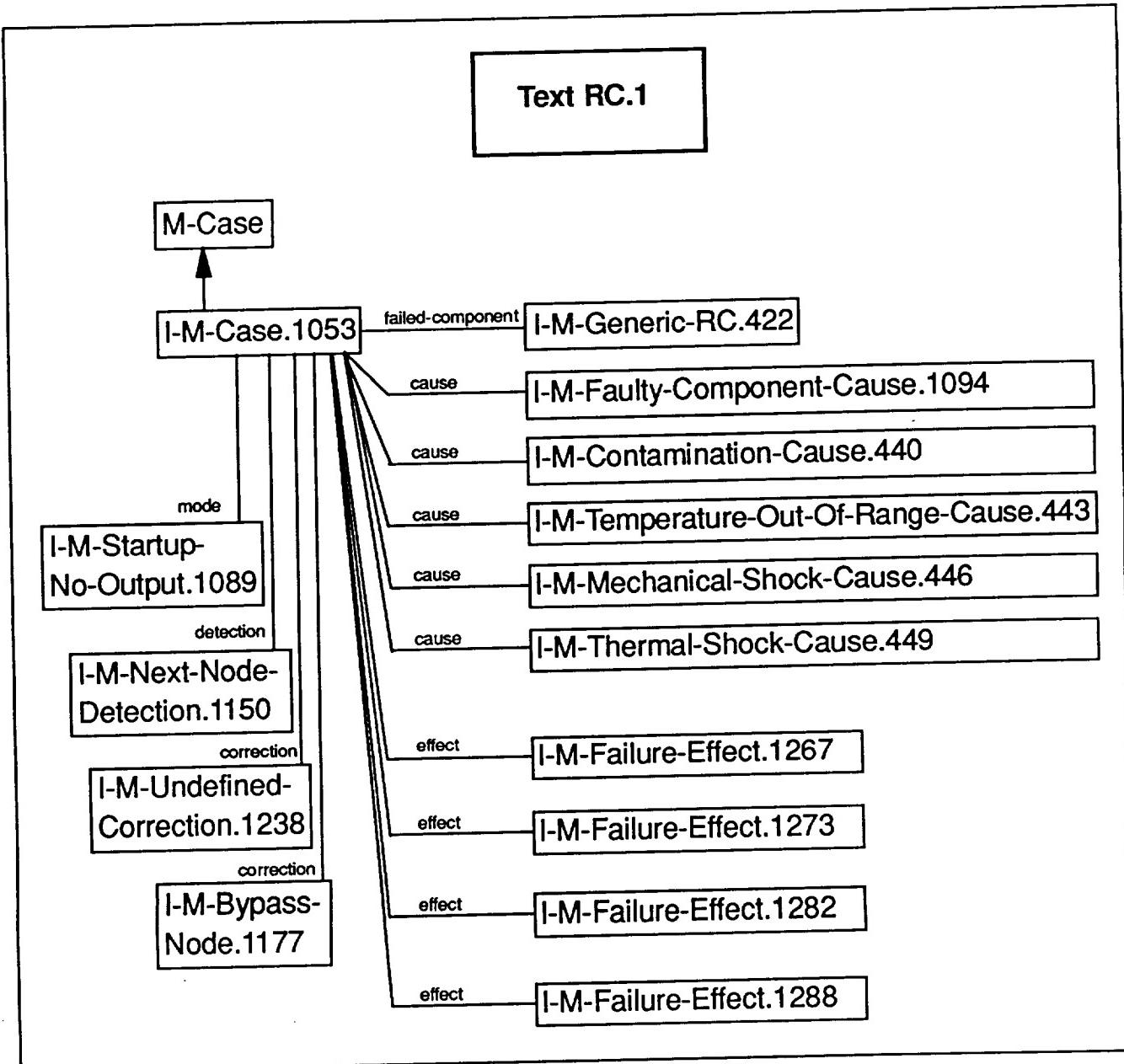


Figure 4.26
Case Representation for RC.1

5. Comprehension of Input Text and Questions

In order to understand input segments of failure analysis manuals, FANSYS requires some means of building a conceptual representation of natural language text in the domain of failure detection, analysis, and repair. This task is performed by the parsing module of FANSYS, which uses case-based parsing techniques (Riesbeck and Martin, 1986; Riesbeck and Schank, 1989) to map input text into the underlying memory structures that characterize the conceptual content of that text. As described in System Architecture (Section 3), failure cases are stored as frames called MOPs within a multi-linked, hierarchical semantic network. This network encodes the DMS fault domain, as described in Domain Knowledge Representation (Section 4). The parser must create a mapping between a piece of text from the Failure Modes Effects Analysis (FMEA) manuals and those MOPs that represent the concepts underlying that text. Such MOPs may already exist within memory, in which case parsing is the recognition of those concepts and relationships in a text that the system has already encountered. Alternatively, new MOPs may be instantiated representing new instances of concepts that the system knows about but has not yet encountered. The parser must also index these MOPs in such a manner as to make them available during future text processing, and during question-answering sessions.

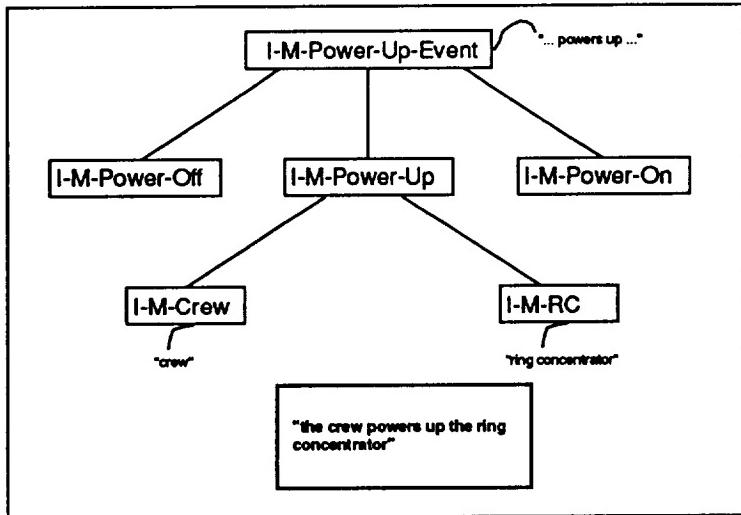


Figure 5.1
Example mapping of text to conceptual representation for the power-up event

To illustrate the process of mapping text into its conceptual representation, consider figure 5.1 which shows the representation of the following sentence fragment: "the crew powers up the ring concentrator." Conceptually, this text fragment describes an event wherein the crew members of the space station apply power to one of the components of the DMS. As described in the Domain Knowledge Representation Section, there are MOPs representing both powering events and systems within the DMS. The MOPs corresponding to those entities from the text fragment would be M-POWER-UP-EVENT, M-CREW, and M-RC. The task of the parser is to recognize each of these events and systems from the text, and to create (or find) the MOPs in memory which encode them. Figure 5.1 suggests the MOP representation that might be referenced by the system for the powering-up text fragment. Note that different pieces of the text map into different segments of the representation, and that there are parts of the representation that are not referenced explicitly in the text. For example, the state transition between the power

being off and the power being on (as represented by the precondition and result slots of I-M-POWER-UP-EVENT) is implicit in the text, and is inferred from domain knowledge.

One approach to creating this mapping between text and its underlying conceptual structures is to specify lexical patterns which capture the stereotypical use of language use. A lexical pattern can form a link to the MOP which characterizes the content of that pattern. For example, a pattern consisting of the words "ring concentrator" might map to the MOP M-RC, since that is how ring concentrators are specified in English. The parser also needs to know that "powering up" (when used in the context of an actor powering up a physical system) maps to the M-POWER-UP-EVENT, and that by virtue of the rules governing English use, the actor of the event will precede the words "powers up" and the object will follow. A general pattern that the system could use to capture this relationship might be "(actor) powers up (object)," where the actor is a concept representing an entity capable of performing a power up action, and the object is a device capable of being powered up. In this scheme, the phrase "ring concentrator" would map straight into M-RC and "the crew" would map into M-CREW. Note that this pattern is general enough to also be used in the following case: "the system management powers up the gateway," with "the system management" mapping into M-SM and "the gateway" mapping into M-GW. The generality of the phrase "(actor) powers up (object)" captures a broad class of mapping instances, which is a reflection of stereotypical language use.

Another point to be made about the parser is its language independence: the conceptual representation mapped into by the parser should be independent of the surface form of the source language. For example, the same MOP built to characterize the sentence "the crew powers on the ring concentrator" should also be created for the sentence "the power system of the RC is activated by the crew members." Processing this sentence entails using a different lexical pattern to capture the notion of an event where an object "is activated by" someone, but the underlying conceptual structures would be the same.

In addition to identifying the memory structures represented within the input text, the parser must perform a number of other tasks to cohesively organize those structures. For example, causal relationships between statements within the text must be recognized (e.g. through application of inferential domain knowledge) and characterized. The salient components of a case description, such as the failure causes, effects, and correction procedures must be recognized and organized for a case instance. Procedures that are composed of multiple steps across several sentences within a text must be constructed and organized. In each of these tasks, domain knowledge, in the form of expectations as to what comprises a procedure, case, or abstract relationship must be applied to the input text to further elucidate those relationships within the text that are often left implicit.

Once the case has been constructed by the parser, it must be stored within memory in a manner which insures that it is available during future parsings of text. This is facilitated by indexing case descriptions with the ISA and slot-filler links described in System Architecture (Section 3). As a case library is built up by the reading of successive texts, generalizations across cases and across the components of a case (procedures, entities, events, etc.) should be drawn. The cases are indexed with the generalization indexing hierarchical scheme to facilitate the generation of such generalizations. This process of drawing these generalizations from text will

be further described in the Memory Search and Retrieval (Section 6). The case library, built incrementally during the processing of manuals, should then be available for question answering tasks. These tasks rely on both the parsing process (to understand natural language queries) and the search and retrieval process (to find answers to the queries).

Since the parser interacts with the Search and Retrieval Module during question-answering sessions, it must work in a tightly coupled manner with memory search and retrieval processes. To this end, the MOPs that are created and recognized by the parser are the same ones used during memory search and retrieval; only the indexing hierarchies and the MOPs created as generalizations are different across the two modules. The natural language question answering shell described in User Interfaces (Section 7) is built directly on top of the Parsing Module. Questions about cases are parsed with the same parser that is used to process cases; functional knowledge associated with the memory structures activated by parsing a question are responsible for generating the query used by the memory search and retrieval processes. This tightly-knit integration of the two modules provides a flexible natural language query environment.

Parsing in the restricted domain of fault detection and correction for the DMS has some ramifications for the parsing process. As figure 2.1 suggests, the text for the FMEA manuals is highly stereotypical in format. Case descriptions are always given with sections for each of the major components of a case, including the causes, effects, correction procedures, detection procedures, and effects. The parser may take advantage of the standard format for a manual to generate expectations about what it will see next in a text. For example, the text following the tag phrase "Correction Procedure:" is likely to be a procedural description of how to correct a failure in a DMS component. These expectations generated after recognizing a word or phrase allow the parser to apply domain knowledge to help process the text at appropriate times.

Another feature of the restricted domain over which the texts deal is the manner in which the task of disambiguation is made easier. The meaning of a word in any natural language can be highly context dependent; words have multiple meanings, and the appropriate meaning is often determined by context. In a restricted domain like that of the DMS, word meanings are highly constrained. For example, the word "network" is likely to deal with a configuration of communicating devices, as in a token ring network, rather than, say, a television network. Likewise the word "ring" might appear as part of the name of a ring concentrator, or as part of a token ring network, but would not be used in the sense of a wedding band. However, domain restrictions do not imply that all words are easily disambiguated within the domain. For instance, the word "failure" is used within many different contexts; disambiguation typically occurs by waiting to see what the subsequent word is, such as in "failure correction" versus "failure mode."

The parser currently parses the case described previously for the failure of the ring concentrator (see figure 2.1). The actual representation for that case was shown in the Domain Knowledge Representation Section (figure 4.26). Both the theoretical and implemented processes used to parse this case will be discussed in the rest of this section, including a detailed, annotated trace of the parser in operation. The processes involved in question-answering and text generation will also be considered. Although only the one case is currently parsed by the system, the same processes developed for the first case are applicable to any of the other cases the parser

may encounter. For example, the first gateway case, GW.1 (loss of output - failure to start) involves the same modes, causes, and effects of the ring concentrator case shown in figure 2.1. The same processes that construct the correction and detection procedures in the ring concentrator will apply to the gateway; only the individual steps that describe the procedure and the type of procedure are different. The methodology used by the parser and other components of the system are designed to generalize across the whole corpus of FMEA manuals, creating a system capable of understanding and operating within the DMS fault domain.

5.1 Case-Based Parsing

In FANSYS, comprehension of input text and questions is performed using the case-based parsing techniques provided by DMAP, a Direct Memory Access Parser (Riesbeck and Martin, 1986; Riesbeck and Schank, 1989). In DMAP, parsing is viewed as a recognition process, i.e., the goal of the parser is to determine which memory structures best organize the input based upon what the parser has already been exposed to. In reading the case describing a gateway failure, the parser will automatically find many of the other case representations it has already seen and use them to help understand the new input. The output of such a parser is the set of memory structures that have been referenced in the understanding of a new text (such as part of some other case), the new structures added to memory during the parse, and the set of expectations about what will be seen next based on what was just read. In this view, parsing is tied directly to the contents of memory; there is no notion of "outputting" some representation. Exactly what occurs when something is referenced in memory is itself guided by the goals of the understanding system, which are also represented in memory. In the case of FANSYS, the goal is to organize the input in ways useful to characterizing the DMS fault domain, and to permit natural language queries over the case library.

Since case-based parsing is seen as a recognition process, memory organization and search are central to any case-based paradigm. MOP structures (Schank, 1982; Riesbeck and Schank, 1989) are used in FANSYS to represent linguistic, domain, and world knowledge. A MOP may be used to represent a word, a concept, or any other semantically useful item within memory. The Domain Knowledge Representation section already detailed several classes of MOPs used to represent domain knowledge within FANSYS. In addition to those MOPs, there are MOPs for each word that the system knows about, as well as MOPs called *index patterns* (Riesbeck and Martin, 1986; Riesbeck and Schank, 1989) which represent stereotypical mappings of natural language into their corresponding concepts. These index patterns drive the parsing process. Other MOPs represent processing knowledge the system has to aid in constructing case descriptions. MOPs are used to implement all knowledge constructs at every level of abstraction and functionality within FANSYS.

MOPs represent nodes in a semantic network, and are organized hierarchically along two parameters that are crucial to the parsing process: the slot-filler hierarchy and the ISA hierarchy. The slot-filler hierarchy links together a MOP with its conceptual subcomponents; for example, the MOP representing an action would organize at least two MOPs in the slot-filler hierarchy: one for the MOP representing the actor, and one for the MOP representing the object being operated upon. MOPs organized within the slot-filler hierarchy thus implement a frame-based representation of knowledge. The ISA hierarchy organizes MOPs based on their conceptual

abstractions, from most general concept to most specific. Thus, the MOP for the power-up action would have as an abstraction the MOP representing power actions in general, which in turn would have an abstraction representing generic actions, and so on. Other specific action MOPs, such as transmission and connection actions would also have the generic action MOP as their abstraction.

MOP-specific processing strategies can also be associated with any given MOP in memory. The processing strategies (called associated functions) are executed whenever a MOP is referenced during parsing or during other processing. Similarly, contextual constraints may be associated with a MOP, which inhibit its activation during a parse if contextual requirements are not met. Thus, domain specific processing strategies, knowledge specific functionality, and contextual requirements are represented and stored with the MOP to which they pertain, and are invoked only when needed. This approach gives the system flexibility in specifying the precise processing characteristics required for different knowledge constructs, and insures that processing proceeds in an integrated manner.

The actual parsing of a text involves the recognition of slot-filler relationships between concepts in a given text. To specify these relationships, FANSYS employs index patterns, which represent stereotypical uses of a target natural language. For example, the pattern { (actor) powers up (object) } => M-POWER-UP-EVENT specifies that a power-up event can be recognized if an actor concept followed immediately by the words "powers up" followed by some object concept is recognized from the input text. The actor and object concepts are slots in the power-up event representation; the index pattern represents the linguistic or conceptual relationship between those slots as expressed in English. Patterns are associated with the MOP that they reference and the MOP with which they are stored by lexical links.

These index patterns also specify lexical and (implicitly) semantic constraints. In the example of the powering-up event, the words describing the actor mop are required to be adjacent to the words "powers up" in order for the pattern to be recognized. Patterns thus implicitly encode syntactic features of a target language. Concepts specified within an index pattern are further constrained by filler restrictions specified within the MOP to which the slot pertains. In the powering-up example, the object in the pattern is constrained to be some mechanical device (e.g. a system) capable of being powered up.

Index patterns are useful for capturing lexical relationships between concepts within text. There are, however, more loosely constrained conceptual relationships which may occur in a text that are not lexically based. In a case description within a FMEA manual, for example, the system needs to know that there are certain components to a case that it should expect within the text: the failure mode, the correction procedures, the effects, etc. These may occur in any order in general (although as mentioned earlier, the FMEA manuals are quite stereotypical in the ordering of the case components -- this may not be the case in other manuals). High level index patterns (HLIPS) exist that capture these kinds of relationships. One example of this type of pattern is { failed-component mode cause detection correction effect } => M-CASE. This pattern indicates that the system should collect any instances of failed components, modes, causes, etc. together that it sees in a text, and use them as slot fillers for a case MOP. These patterns describe the high level organization of a target text. One characteristic of these patterns that contrasts to lexical

index patterns is that all components of the pattern need not be present in the text; the HLIPs just encode predictions about what may or may not be seen in the text.

In addition to lexical patterns and HLIPs, other types of processing strategies are needed to apply domain knowledge in building a coherent representation of a case. These types of strategies implement a form of inference generation. Inferences need to be drawn in many cases because much is often left implicit in a text that needs to be made explicit in order for the system to tie together relationships and concepts within a text. Consider for example the sentence: "Indication of a RC failure is first detected by the next active node on the network." The phrase "next active node on the network" is enough to recognize the relationship of two functioning nodes on the network, one of which is in some sense the next node to the other (in the sense of the ordered passing of a token in a token ring network, for example). The fact that the next node is itself a ring concentrator, and that it is waiting for a token from the failed node, are all left implicit, and must be inferred from the understander's domain knowledge about token ring networks.

Another type of inferential strategy is that which pulls together steps from a procedures, determines which procedure is being referenced, and then recognizes that procedure. Consider the following correction procedure from the first case description: "The crewmen check for applied power and check for connector tightness. If the RC cannot then be placed in operation, it is removed and replaced with an ORU logistics spare." The fragment details essentially three steps to a procedure: checking for power, checking the connectors, and replacing a unit with a spare. These three steps form a procedure for correcting a failure. The parser must be able to recognize that these events are steps in a procedure, and then to recognize the procedure in memory that is being referenced. In this case, it is the procedure called M-NON-OPERATIONAL-REPLACEMENT, described earlier in figure 4.24.

These inferential strategies are generally attached to specific nodes in memory that get referenced during a parse. For example, the procedure building strategy is attached to the MOP for an abstract event, so that any event that is referenced during a parse will execute it. Inferences are thus localized to the MOPs to which they apply, and are activated naturally in the course of the recognition of those MOPs. The mechanisms for implementing these strategies, and for the parsing of text in general, are described next. An annotated trace will also exemplify these processes.

5.2 Parsing Input Text

As mentioned in the previous section, the stereotypical use of natural language is captured through the use of lexical patterns (called index patterns) which map text into the memory structures underlying the conceptual content of that text. Index patterns are themselves MOPs, indexed via lexical activation links with the MOP that is most predictive of the pattern, and with the MOP to which the pattern refers via lexical reference links. In the example of IP1: { (actor M-SYSTEM) powers up (object M-ENTITY) } => M-POWER-UP-EVENT, the pattern would be stored with the lexical MOP for the word "powers," since that is the most relevant element in the pattern to its recognition. The pattern should not be stored with the M-SYSTEM MOP (which is the filler for the actor slot), for example, because actors appear in many different

patterns and contexts. IP1 would also have a link to the MOP M-POWER-UP-EVENT, since that is the MOP characterized by the pattern. If a MOP is referenced in the text or through subsequent processing, any index patterns that are associated with that node are activated. For example, anytime the MOP for the word "powers" is referenced in the text, pattern IP1 becomes activated. If the elements that appear prior to that MOP in the pattern have occurred in the text in the proper lexical order (i.e. lexical adjacency requirements are met), then the pattern is considered active and will be tracked. In the example if IP1, the M-SYSTEM MOP would have had to have been referenced just prior to the MOP for the word "powers" in the previous pattern. Once a pattern is tracked, a prediction is generated about what will appear next in the text based on that pattern. To continue the example, the parser would predict that the MOP for the word "up" will appear next in the text. If this prediction fails, the pattern is abandoned. If the prediction is confirmed, then the pattern will be further advanced, and the next element of the phrase predicted (in the example of IP1, it would be the node representing the object slot filler, M-ENTITY). When all of the elements of a pattern have been recognized in the text, the target of the pattern is recognized with its slots filled based on the text. In the example "the crew powers up the ring concentrator," an instance of the M-POWER-UP-EVENT MOP would be instantiated with the slots of (actor I-M-CREW) and (object I-M-GENERIC-RC).

The application of index patterns involves a mixture of bottom-up and top-down processing. Bottom-up processing occurs in the recognition of lexical units (words, usually) from the input text. These words may trigger the activation of index patterns, which then spawn predictions about what will be seen next in the text. Subsequent words or concepts may fulfill top-down expectations as generated by index patterns, as in the prediction of the word "up" or the concept of M-ENTITY in IP1. Patterns which are fully recognized lead to the activation of new MOPs, which may themselves have index patterns associated with them. The output of the parser after reading a text is thus the memory structures that have been created or recognized to organize the input, and the new predictions that are pending based on that reading.

Index patterns are defined as in figure 5.2. The parts of the definition in square brackets are optional. The syntax-category attribute is a simple method of encoding syntactic constraints to be used during generation. (See the section on Generation for more details on the use of syntactic categories; they do not effect parsing.) The target is the MOP to be activated if all of the elements of a phrase are recognized; the target thus specifies the lexical activation link. Elements (elem1 ... elemn) may be either words (lexical nodes) or concepts in memory. The :terminator specifies when the pattern should be destroyed if it hasn't been recognized. This is typically set to be M-HARD-PUNCTUATION (e.g. when a period is parsed), and is used in the cleaning up of abandoned patterns. The :store node specifies which of the elements of the pattern should be used as the lexical activation link; that is, which element is most predictive of the pattern. The :gen and :nogen fields are used during generation; see the Generation subsection for more details, as they do not influence parsing directly. Finally, the :opt allows the system to specify optional elements of a pattern. These elements need not appear in the text in order for the whole pattern to be recognized. If they are in the text, they are parsed as part of the pattern, but they are ignored otherwise. The definition specifiers given in figure 5.2 allow index patterns to be defined with a fair degree of generality. See appendix C for a listing of the pattern definitions used by the system.

```
(defphrase [syntax-category] target elem1 [elem2 ... elem n]
          [:termination termmop] [:start elem-number] [:gen] [:nogen] [:opt optlist])
```

Figure 5.2
Index Pattern Structure

In addition to index patterns, there is also a class of closely related structures called higher-level index patterns (HLIPS). These are similar to normal index patterns, except that they do not require that any adjacency requirements be met. HLIPS typically capture relationships that occur across large sections of a text that are not order dependent. For example, case descriptions have a number of features including the correction and detection procedures, the failure causes and modes, the failure effects, and so on, that could in principle appear in any order within a manual description. There is thus a HLIP defined that looks for instances of each of those features and collects them together to create a case MOP. Appendix C also details those patterns as used to parse the first ring concentrator case.

5.2.1 Marker Passing

Mention has been made of the generation and fulfillment of predictions from index patterns, as well as the recognition (or activation) of structures within memory. These features have been implemented in the FANSYS parser using a marker passing algorithm suggested (but not implemented) in microversion of DMAP appearing in (Riesbeck and Schank, 1989). Marker passing may be defined as the propagation of structured objects over some representational network. The intersection of markers during this propagation typically allows useful work to be performed. (See (Hendler, 1988) and (Norvig, 1989) for an introduction to marker passing methods in general.)

In FANSYS, two types of markers are propagated: prediction markers and activation markers. Prediction markers are passed to nodes in memory which are predicted by advancing index patterns. These markers contain backpointers to the index pattern (or other MOP processing structure) which generated it. Activation markers are passed to MOPs and their abstractions which have been referenced in input text, or have been activated by other means (e.g. by activating the target node of a recognized phrase). The collision of these two types of markers at a MOP in memory represents a fulfilled expectation. The node that generated the prediction is subsequently informed of fact that the prediction was satisfied. Activation markers thus keep track of the state of activation within memory; that is, which MOPs have been recognized during the parsing of text. Prediction markers keep track of the state of expectations within the system. Together, these markers implement an efficient scheme for generating and fulfilling predictions within FANSYS.

Figure 5.3 shows the structure of an activation marker. The parent of an activation marker is a backpointer to the node which was initially activated (since markers may be passed around the ISA hierarchy, a backpointer to the original node is necessary). The owners attribute is a list of all of the MOPs which have been passed the activation marker; this is just a device to aid in the eventual removal of the markers from the system. The start and finish attributes are used by index patterns to enforce adjacency requirements. When an index pattern is first

instantiated, it is given the "start" value of a counter which is incremented by one every time a word is read in the text. When the pattern is fulfilled, the value of "finish" is set to the current value of the counter. When a node is referenced by an index pattern, it in turn inherits the start and finish values of the pattern for its activation marker. These values are then used within other index patterns to enforce adjacency of concepts and words in the pattern. Adjacency may be determined by noting that two nodes are adjacent if and only if the finish attribute on the marker of the first node is one less than the start attribute of the marker on the second node.

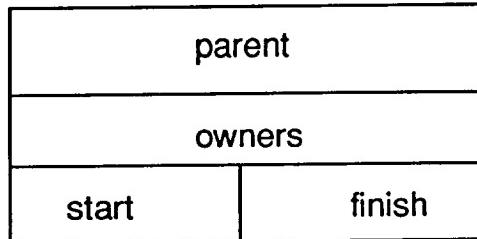


Figure 5.3
Activation Marker Structure

The structure of a prediction marker is demonstrated by figure 5.4. The only information stored with a prediction marker is a backpointer to the MOP which spawned the prediction. This pointer is used to inform the predicting node of the fact that the prediction was fulfilled.

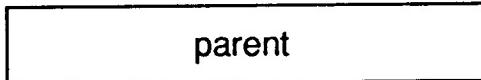


Figure 5.4
Prediction Marker Structure

The basic marker passing algorithm which implements the prediction/activation mechanism is an adaptation of the one presented in (Riesbeck and Schank, 1989). It consists of the following four rules for passing markers:

Activation Markers

- When a word is read from the input text, an activation marker is given to the corresponding lexical node.
- When every element of a phrase receives an activation marker, the target concept attached to the phrase receives a marker.
- When a node receives a marker, it sends copies to all of its abstractions up the ISA links.

Prediction Markers

- When an element of a phrase in an index pattern receives an activation marker, the next element in the phrase gets a prediction marker.

The collision of an activation and a prediction marker causes the parent node of the prediction to be activated. Associated functions for that MOP may then guide processing to deal with the fulfilled expectation. For example, an index pattern node that is activated by a fulfilled prediction has functional knowledge that would advance the pattern one element, filling in any appropriate slots, or that would activate the target node if all elements had been recognized.

One final note concerning the marker passing algorithm is worth making. This marker passing scheme is independent of the configuration of memory, and thus is independent of the domain in which it operates and source language with which it is concerned. It is a general mechanism for implementing activation, expectations, and fulfilled predictions. Note on the other hand that the actual comprehension of text is dependent on the memory configuration, since different memory structures and domain knowledge will cause a different pattern of activation throughout memory.

5.2.2 The Parsing Algorithm

The actual process of parsing text into the internal representation that characterizes it is founded upon a simple algorithm:

1. Get the next word from the text.
2. a) If a lexical node exists in memory corresponding to that word, activate that node.
- b) Otherwise, create a lexical node for that word and index that MOP under the MOP M-UNKNOWN-WORD.

Each word that FANSYS knows has a corresponding MOP in memory. Words are parsed from left to right, with each MOP corresponding to a word being activated when that word is read (the concept of activation will be discussed in a moment). Words that are not recognized are added to memory as lexical nodes, but are indexed as unknown words. This allows the administrator of the system to notice new words and add any corresponding processing knowledge to the system to handle that word. (In future implementations of the system, it may be possible to have the system implement learning strategies for unknown words automatically, based on the context in which the word occurs; such strategies could be stored as functional knowledge with the MOP M-UNKNOWN-WORD.)

Activation is the fundamental memory operation for the parser. A node is activated if it has some relevance to the processing of the input text. (The notions of the recognition of a MOP and the activation of a MOP are used interchangeably within this report.) The activation operation consists of the following six steps:

1. The MOP is specialized to an appropriate instance, if there are any specific slots passed to the activation function. This may involve finding a prior instance in memory that matches the MOP and its slots, or it may involve creating a new instance of the MOP in memory.
2. An activation marker is created for the node. If the node is activated with an activation marker passed to it, it gets a copy of that marker. Otherwise, a new marker is created for this node.
3. Any functions associated with that node are executed. These functions typically encode the structure-specific (semantic) content of that node (i.e. how to process it).
4. All of the index patterns which are stored with this MOP via lexical activation links are activated.
5. If this node was predicted (because prediction markers were passed to it previously), then the predictions are fulfilled by activating the MOP or MOPs that predicted it.
6. All of the abstractions of this node are activated, and are passed this node's activation marker.

Whenever a concept in memory is activated, it is specialized to the most specific instance of that MOP that matches the input. Typically, an abstract MOP is activated with a set of slots and fillers specified. Consider the example presented earlier, where the M-POWER-UP-EVENT MOP is activated with the slots of (actor I-M-CREW) and (object I-M-GENERIC-RC). If the parser has already seen an instance of M-POWER-EVENT with the specified slots, it would find that instance and activate it. Otherwise, the parser is seeing novel text, and must create a new instance of the MOP with the appropriate slots. (This new instance may cause some reorganization of memory, since the Memory Creation and Retrieval module may use it to draw generalizations about the text; see the section on Memory Creation and Retrieval for more information.) This process of specialization demonstrates how the parser uses a case-based approach: new input text is mapped onto already existing (i.e. previously experienced) memory structures, if such structures are available. Memory content (experience) guides parsing. Since new MOPs are built during parsing if no suitable MOPs exist in memory, parsing changes memory. If a text is read twice, most of the processing (the creation of all new MOPs) occurs during the first reading; the second time around, the text is mapped onto the same structures used or created during the first reading. Thus, FANSYS learns from reading a text, making subsequent processing of similar material more efficient.

When a MOP is activated, any functions associated with that MOP are executed. This allows MOP-specific processing strategies to be executed only as needed. For example, the code that handles the processing of index patterns is stored with the abstract node M-INDEX-PATTERN. Whenever an index pattern MOP is activated, this code gets executed, and the pattern is processed. Thus, the parsing process involves domain- and MOP-specific processing in addition to the domain-independent processes described in the previous algorithms. FANSYS' parser can therefore be viewed as a distributed parser, since domain and knowledge specific parsing strategies are stored throughout memory as needed, and come into play only when such associated MOPs are activated. This also provides an adaptable and extendible environment,

since new domains can be added to the model by defining the MOPs required for that domain, and by specifying any special processing requirements with those MOPs.

Another event which occurs when a MOP is activated is the activation of all of that MOP's abstractions. This captures the notion that when one is talking about M-POWER-UP-EVENT, one is also talking about powering events in general (e.g. M-POWER-EVENT), as well as generic events (M-EVENT). Since index patterns can define textual relationships at varying levels of abstraction, this mechanism allows the parsing of input text to also occur on multiple levels of generality simultaneously. For example, the M-POWER-UP event may have index patterns associated with it, as may the M-EVENT MOP; these patterns would vary in terms of their generality, but each would become activated when a power down event is processed in the text.

Since the activation of a MOP usually leads to the subsequent activation of several other MOPs, including abstractions, index patterns, predicting nodes, and so on, the parsing of a word typically involves the spreading of activation over several parts of the memory of FANSYS. This spreading activation is constrained by the algorithm, but allows processing to flow to those structures which are required to process the text. This provides an efficient and domain-independent method for understanding text, but one which is still guided by the actual content of the domain representation.

5.2.3 Domain-Specific Inference Strategies

In addition to the normal flow of control detailed by the parsing algorithm and the tracking of index patterns, some domain specific inferencing strategies are implemented which also effect the processing of an input text. These inferencing strategies are generally attached to specific processing MOPs as associated functions. Such processing MOPs may become activated by higher level patterns or through explicit activation by other associated functions. These strategies represent domain-specific parsing knowledge, and are inferential in that they typically involve processing that makes relationships and concepts that are implicit within a text explicit.

One example of such an inferencing strategy is the functionality which constructs and recognizes procedures within textual descriptions. Procedures occur within the text in two primary contexts, that of the detection procedure and the correction procedure. Procedures are usually specified in one of two ways (as has been determined by a study the input manuals). In the first case, a pointer to the class of procedure is given as the first sentence of the textual description, followed by one or more events which serve to disambiguate which instance of that class is being recognized. For example, consider the failure detection procedure of the first ring concentrator case, RC.1 (see figure 2.1). The first sentence of this description, "Indication of a RC failure is first detected by the next active node on the network," is a pointer to the class of detection procedures in which a failure is detected by another node noticing that some expected communication with it has failed to occur. The next sentence, "Local system management will reach a time-out limit for receipt of the network token," is an event which disambiguates the detection procedure. In this case, since the event details a token circulating around a token ring which fails to be passed to a node, the parser may determine that the procedure in question is the M-NEXT-NODE-DETECTION procedure.

The second class of procedural recognition strategies occurs when a text elaborates two or more steps of a procedure. In this case, the parser must recognize each step, pull them together, and infer which procedure in memory is being referenced by the text. As an example of this class, consider the long-term corrective procedure of RC.1. In this example, three steps are mentioned in the text: checking for applied power, checking for connector tightness, and replacing the spare. These are three steps of the procedure M-NON-OPERATIONAL-REPLACEMENT, which must be recognized and activated during the parse of RC.1.

Each of these two cases requires a specific strategy implementation to determine the procedure being referenced (and thus to ultimately activate that procedure in memory), although the fundamental functionality is the same in both cases. In the first case, a pointer is given to the class of procedures involved, which significantly reduces the space of procedures which must be searched for the target procedure. In the second case, the library of all procedures (correction or detection, depending on context) must be searched. Once the search space is determined by the strategy type, the next step is to collect together the subsequent events in the text, and determine which procedure they comprise. It should be noted that usually not all of the events in a procedure are specified in a text; many are left implicit, with the assumption that the reader's domain knowledge will be sufficient to fill in the missing details. Since all of the events must be collected together, a processing MOP is activated by the first event to occur in the text which will keep track of all of the events that are subsequently referenced (this activation occurs by virtue of an associated function attached to M-EVENT). This MOP subsequently predicts that it will see more events by passing a prediction marker to M-EVENT each time it is activated. If that prediction is fulfilled, the processing MOP will again be activated, and it can deal with the new event, and predict that it will see yet more events. Once the procedural context is left by parsing a new section of the text, the processing MOP considers all of the events it collected and determines which procedure was referenced.

The algorithm for determining which procedure is to be inferred from the collected events is a simple one. Each event step contains a back pointer to each procedure in which it occurs. (In fact, all MOPs contain a backpointer to any other MOPs in which it occurs as a slot filler.) Since an event may occur in more than one procedure, there may be several such backpointers. Once all of the steps are collected together, the processing MOP intersects all of the backpointers together from all of the events; this intersection is likely to be just one MOP, the procedure which packages all of the steps. If more than one MOP is in the intersection set, than some other method may be required to further disambiguate the procedure. As our corpus of procedures is limited by the number of cases that are dealt with, this has not proven necessary -- the process of intersecting the backpointers has proven sufficient to uniquely determine all procedures tested. It is recognized however that with a more extensive library of procedure descriptions, a more generalized approach to procedural referencing may be necessary. Once the procedure has been identified by whatever means, it is activated and the processing MOP created for the recognition task is terminated.

There are several other miscellaneous inferencing strategies that are employed to process the text for RC.1 (see figure 2.1). One example of this is the disambiguation of the word "none" in the failure effects procedures. There are two practical meanings of the word "none" in this context. The first is exemplified by the failure effects on the crew in RC.1. Although the effect is

listed as none, there actually is one effect on the crew; they must implement the correction procedure described by the text after the word "none". This text is almost always a pointer back to one of the correction procedures described earlier in the case. The task of the parser here is to infer that the real effect is the correction procedure described next in the text. The inferencing function (associated with the word "none" when it appears in the context of the effects section of the text) must therefore search back over the correction procedures to find the one which best matches the text in the given effects text. In the case of RC.1, the phrase "the conditions associated with the replacement of the RC" is enough to disambiguate between the two correction procedures, since the procedure M-NON-OPERATIONAL-REPLACEMENT is the one that involves an event with the crew replacing a spare RC. For the other effects, where there is no subsequent text after the word "none", the parser must infer that the default procedure M-NULL-PROCEDURE is the actual effect (i.e. that there is indeed no effect). This is the second practical meaning of "none" in this context.

Some other inferencing procedures implement contextual constraints and miscellaneous disambiguation functions. These will not be described here in any detail. The interested reader is invited to examine the code for more details. It should be noted that although most of the inferencing strategies are founded upon a generalized theory of what should constitute a reasonable inference, it is not claimed that all of the inferencing strategies implemented to parse the first case are of a general nature. Some strategies will undoubtedly generalize as more cases are processed by the system.

5.2.4 An Annotated Example

To illustrate the processes that have been discussed in the prior subsections, consider the actual trace output of the parser as it parses the first ring concentrator case, RC.1. The trace is highly edited due to spatial considerations; only features which illustrate relevant processing will be left intact. Interested readers should consult Appendix B for a complete copy of the trace. Note also that details will be omitted once they have been covered in the trace; this will have the effect of creating a sparser trace as this subsection progresses.

>(parse rc.1)

Parsing (ITEM NAME *COLON* RING CONCENTRATOR FAILURE MODE *COLON* LOSS
OF OUTPUT - FAILURE TO START FAILURE CAUSES *COLON*
PIECE-PART FAILURES *COMMA* CONTAMINATION *COMMA*
TEMPERATURE *LEFT-PAREN* HIGH OR LOW *RIGHT-PAREN*
COMMA MECHANICAL SHOCK *COMMA* THERMAL SHOCK FAILURE
DETECTION *SLASH* VERIFICATION *COLON* INDICATION OF A RC
FAILURE IS FIRST DETECTED BY THE *QUOTE* NEXT *QUOTE*
ACTIVE NODE ON THE NETWORK *PERIOD* SYSTEM MANAGEMENT
WILL REACH A TIME-OUT LIMIT FOR RECEIPT OF THE NETWORK
TOKEN *PERIOD* CORRECTIVE ACTION *COLON* *LEFT-PAREN* A
RIGHT-PAREN SHORT TERM *COLON* NETWORK RECONFIGURATION
IS EFFECTED AUTOMATICALLY *PERIOD* THE DMS NETWORK
REMAINS IN OPERATION IN A RECONFIGURED STATE WITH THE
FAILED RC BYPASSED *PERIOD* *LEFT-PAREN* B *RIGHT-PAREN*
LONG TERM *COLON* THE CREWMEN CHECK FOR *QUOTE* APPLIED
POWER *QUOTE* AND THE CREWMEN CHECK FOR *QUOTE* CONNECTOR

TIGHTNESS *QUOTE* *PERIOD* IF THE RC CANNOT THEN BE PLACED IN OPERATION *COMMA* THE RC IS REMOVED AND REPLACED WITH AN ORU LOGISTICS SPARE *PERIOD* FAILURE EFFECT ON *COLON* *LEFT-PAREN* A *RIGHT-PAREN* CREW *SLASH* SSPE *COLON* NONE *PERIOD* THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD* *LEFT-PAREN* B *RIGHT-PAREN* MISSION SUPPORT *COLON* NONE *PERIOD* *LEFT-PAREN* C *RIGHT-PAREN* SYSTEM *COLON* NONE *PERIOD* *LEFT-PAREN* D *RIGHT-PAREN* INTERFACES *COLON* NONE *PERIOD* *EOC*).

The parser begins by processing the text in a word-for-word manner. The first word, "item", is activated as described in the activation section. This includes activating the word, its abstractions, any index patterns associated with it, and so on.

Reading ITEM

Recognizing: M-IP.388
Creating: I-M-IP.388.417
Specializing: I-M-IP.388.417
Recognizing: I-M-TERMINATOR.323
(ITEM * NAME *COLON*) = M-ITEM-CONTEXT

In activating the word "item", one index pattern has been found which is stored with that word. This pattern is activated. As usual in the activation process, the concept is recognized, and an instance is created for it. FANSYS then determines whether it has seen this input before by trying to specialize the input into an existing memory structure. If it is successful, the parser will throw away the new instance and activate the prior occurrence. Otherwise, it will store this new instance in memory. In this case, the index pattern is a new one, and so is added to memory. The I-M-TERMINATOR.323 MOP is a bookkeeping MOP which controls when this pattern will be killed. The designer of the lexical phrases may specify when a given phrase should be terminated; the default is at the end of a sentence. The MOP M-ITEM-CONTEXT is a MOP which specifies that the current context is that of the failure item. Within this context, any devices that are specified are interpreted as the failed component(s) of the case. In order for this context to be activated, the full tag of "item name:" must be encountered in the text. The model is thus relying on the stereotypical format of the case manuals to determine the context in which it is processing. Notice that if the phrase "item name:" is encountered anywhere within the text, the parser will activate this context. This has the potential of confusing the system, but the manuals are generally structured such that this is not a problem.

Reading NAME

(ITEM NAME *COLON*) = M-ITEM-CONTEXT

Here the pattern continues to be tracked by the system.

Reading *COLON*
(ITEM NAME *COLON* *) = M-ITEM-CONTEXT referenced
Recognizing: M-ITEM-CONTEXT
Creating: I-M-ITEM-CONTEXT.418
Specializing: I-M-ITEM-CONTEXT.418

Recognizing: M-CONTEXT
Creating: I-M-CONTEXT.420
Specializing: I-M-CONTEXT.420
Removing: I-M-IP.388.417

Since the whole phrase has been recognized, the target of the phrase, M-ITEM-CONTEXT is activated. The functional knowledge associated with this MOP activates the general M-CONTEXT MOP, which is tasked with the responsibility of maintaining the current context. The M-CONTEXT MOP in turn makes note of the new context. This context inhibits the activation of some higher level patterns and allows the activation of others. Specifically, it will allow the activation of those high level patterns which keep track of the failed-component slot for a case, and inhibit the activation of patterns that are used in the other contexts. The last bit of processing involves removing the fulfilled index pattern from the system.

Reading RING
(RING * CONCENTRATOR) = M-GENERIC-RC

For index patterns, the recognize/create/specialize/termination sequence will be edited out from here on and only the pattern will be shown as it is tracked.

Reading CONCENTRATOR
(RING CONCENTRATOR *) = M-GENERIC-RC referenced
Recognizing: M-GENERIC-RC
Creating: I-M-GENERIC-RC.422
Specializing: I-M-GENERIC-RC.422
Recognizing: M-HL-PATTERN.387
Creating: I-M-HL-PATTERN.387.423
Specializing: I-M-HL-PATTERN.387.423
Removing: I-M-IP.327.421

With this word, the pattern recognizing a generic instance of a ring concentrator is fulfilled, and that instance is activated. This causes a high level pattern to be activated; this pattern collects any references to ORUs within this context, so that the appropriate failed-component slots may be added to the case.

Reading FAILURE

Note that no phrases are associated with the word failure, since it is not particularly predictive in this domain.

Reading MODE
(FAILURE MODE * "COLON") = M-MODE-CONTEXT

Here is an example of a phrase that is stored with a concept other than the first element of the phrase. In this case, the parser checks to verify that the concepts in the phrase prior to the word "mode" are present in the proper lexical order. This is accomplished by checking the activation markers associated with the words or concepts in the phrase to see if they were activated at the appropriate time. Had they not been, the phrase would be terminated at this point. This particular phrase survives the test, and so is tracked. Note that it is a phrase that may establish a new parsing context.

Reading *COLON*

(FAILURE MODE *COLON* *) = M-MODE-CONTEXT referenced

Specializing: I-M-MODE-CONTEXT.425

Removing: I-M-CONTEXT.420

Specializing: I-M-CONTEXT.427

Removing: I-M-IP.389.424

The recognition/creation of ordinary concepts is now abbreviated from this point on. The M-MODE-CONTEXT establishes a new context for the parser; there is now an expectation that failure modes will be processed next. With this new context, the old context is destroyed. At this point, the failure components that have been collected together (in this case, just M-GENERIC-RC) are added to the case (this is not reflected in the trace). Every time a new context is encountered, any processing in the prior context is resolved and terminated. This typically involves the addition of new slots to the case description. It is necessary to wait for the new context to finish processing, since the parser has no other means of determining when a section of the text is finished (people might rely on white space between sections, for example).

Reading LOSS

(LOSS * OF OUTPUT - FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

(LOSS * OF OUTPUT - FAILURE TO START) = M-STARTUP-NO-OUTPUT

Two phrases are attached to the word "loss". Both will be tracked simultaneously until the patterns are either fulfilled, or terminated due to a failed lexical prediction.

Reading OF

(LOSS OF * OUTPUT - FAILURE TO START) = M-STARTUP-NO-OUTPUT

(LOSS OF * OUTPUT - FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

Reading OUTPUT

(LOSS OF OUTPUT * - FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

(LOSS OF OUTPUT * - FAILURE TO START) = M-STARTUP-NO-OUTPUT

Reading -

(LOSS OF OUTPUT - * FAILURE TO START) = M-STARTUP-NO-OUTPUT

(LOSS OF OUTPUT - * FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

Reading FAILURE

(LOSS OF OUTPUT - FAILURE * DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

(LOSS OF OUTPUT - FAILURE * TO START) = M-STARTUP-NO-OUTPUT

Reading TO

(LOSS OF OUTPUT - FAILURE TO * START) = M-STARTUP-NO-OUTPUT

At this point, one of the phrases fails to advance because of a failed lexical prediction. This phrase is no longer tracked by the system and will be removed at the appropriate time (i.e. when the end of a sentence is encountered).

Reading START

(LOSS OF OUTPUT - FAILURE TO START *) = M-STARTUP-NO-OUTPUT referenced

Specializing: I-M-STARTUP-NO-OUTPUT.430

Specializing: I-M-HL-PATTERN.387.431

Removing: I-M-IP.370.429

A high level pattern collecting together failure modes is now activated. At this point in the trace, the removal of index patterns will be omitted, except in noteworthy circumstances.

Reading FAILURE

Reading CAUSES

(FAILURE CAUSES * *COLON*) = M-CAUSE-CONTEXT

Reading *COLON*

(FAILURE CAUSES *COLON* *) = M-CAUSE-CONTEXT referenced

Specializing: I-M-CAUSE-CONTEXT.433

Specializing: I-M-CONTEXT.435

A new context is established in which the parser has expectations that failure causes will be processed next. The processing for failure modes is now wrapped up, with a slot for the failure mode I-M-STARTUP-NO-OUTPUT.430 being added to a case (again, this is not reflected in the trace).

Reading PIECE-PART

(PIECE-PART * FAILURES) = M-FAULTY-COMPONENT-CAUSE

Reading FAILURES

(PIECE-PART FAILURES *) = M-FAULTY-COMPONENT-CAUSE referenced

Specializing: I-M-FAULTY-COMPONENT-CAUSE.437

Specializing: I-M-HL-PATTERN.387.438

I-M-HL-PATTERN.387.438 is a pattern which collects together failure causes.

Reading *COMMA*

Reading CONTAMINATION

(CONTAMINATION *) = M-CONTAMINATION-CAUSE referenced

Specializing: I-M-CONTAMINATION-CAUSE.440

Specializing: I-M-HL-PATTERN.387.441

Note that "contamination" is a fairly common word, and it activates the MOP M-CONTAMINATION-CAUSE each time it is encountered. However, such activation is irrelevant in any context outside of the failure cause context, since the high level pattern which collects such causes is inhibited in any other context. If this were in any context except the failure cause context, this would be an example in which something is activated within the system which causes no useful processing to occur.

Reading *COMMA*

Reading TEMPERATURE

(TEMPERATURE * "LEFT-PAREN" HIGH OR LOW "RIGHT-PAREN") =
M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading *LEFT-PAREN*

(TEMPERATURE *LEFT-PAREN* * HIGH OR LOW *RIGHT-PAREN*) =
M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading HIGH

(TEMPERATURE *LEFT-PAREN* HIGH * OR LOW *RIGHT-PAREN*) =
M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading OR

(TEMPERATURE *LEFT-PAREN* HIGH OR * LOW *RIGHT-PAREN*) =
M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading LOW

(TEMPERATURE *LEFT-PAREN* HIGH OR LOW * "RIGHT-PAREN") =
M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading *RIGHT-PAREN*

(TEMPERATURE *LEFT-PAREN* HIGH OR LOW *RIGHT-PAREN* *) =

M-TEMPERATURE-OUT-OF-RANGE-CAUSE referenced

Specializing: I-M-TEMPERATURE-OUT-OF-RANGE-CAUSE.443

Specializing: I-M-HL-PATTERN.387.444

Note that parentheses are not handled in any general sense. They are merely incorporated within specific phrases. Since the text in the manuals is so stereotypical in format, this approach handles most cases that are encountered. A more general approach to processing parentheses remains to be developed.

Reading *COMMA*

Reading MECHANICAL

(MECHANICAL * SHOCK) = M-MECHANICAL-SHOCK-CAUSE

Reading SHOCK

(MECHANICAL SHOCK *) = M-MECHANICAL-SHOCK-CAUSE referenced

Specializing: I-M-MECHANICAL-SHOCK-CAUSE.446

Specializing: I-M-HL-PATTERN.387.447

Reading *COMMA*

Reading THERMAL

(THERMAL * SHOCK) = M-THERMAL-SHOCK-CAUSE

Reading SHOCK

(THERMAL SHOCK *) = M-THERMAL-SHOCK-CAUSE referenced

Specializing: I-M-THERMAL-SHOCK-CAUSE.449

Specializing: I-M-HL-PATTERN.387.450

Reading FAILURE

Reading DETECTION

(FAILURE DETECTION * "SLASH" VERIFICATION "COLON"
 (SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP)) = M-FAILURE-DETECTION
(FAILURE DETECTION * "SLASH" VERIFICATION "COLON") = M-DETECTION-CONTEXT
(DETECTION * PROCEDURE "COLON"
 (SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP)) = M-FAILURE-DETECTION

Reading *SLASH*

(FAILURE DETECTION "SLASH" * VERIFICATION "COLON") = M-DETECTION-CONTEXT
(FAILURE DETECTION "SLASH" * VERIFICATION "COLON"
 (SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP)) = M-FAILURE-DETECTION

Reading VERIFICATION

(FAILURE DETECTION "SLASH" VERIFICATION * "COLON"
 (SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP)) = M-FAILURE-DETECTION
(FAILURE DETECTION "SLASH" VERIFICATION * "COLON") = M-DETECTION-CONTEXT

Reading *COLON*

(FAILURE DETECTION "SLASH" VERIFICATION "COLON" *) =
 M-DETECTION-CONTEXT referenced
Specializing: I-M-DETECTION-CONTEXT.454
Specializing: I-M-CONTEXT.456
(FAILURE DETECTION "SLASH" VERIFICATION "COLON" *
 (SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP)) = M-FAILURE-DETECTION

A new context is created for detection procedures. At this point, the failure causes have been added to the case representation.

Reading INDICATION

(INDICATION * OF A (FAILED-COMPONENT M-ORU) FAILURE IS FIRST
 DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) =
 M-NEXT-NODE-DETECTION

Reading OF

(INDICATION OF * A (FAILED-COMPONENT M-ORU) FAILURE IS FIRST
 DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) = M-NEXT-NODE-DETECTION

Reading A

(*LEFT-PAREN* A * "RIGHT-PAREN") = M ENUMERATION
(INDICATION OF A * (FAILED-COMPONENT M-ORU) FAILURE IS FIRST
 DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) =
 M-NEXT-NODE-DETECTION

Reading RC

(RC *) = M-RC referenced
Specializing: I-M-RC.460
(INDICATION OF A (FAILED-COMPONENT M-ORU) * FAILURE IS FIRST
 DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) =
 M-NEXT-NODE-DETECTION

The ring concentrator fulfills the requirement of finding an ORU at this point in the phrase, and so it is advanced. If and when M-NEXT-NODE-DETECTION is recognized, it will be instantiated with the slot and filler (FAILED-COMPONENT I-M-RC.460).

Reading FAILURE

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE * IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) =
M-NEXT-NODE-DETECTION

Reading IS

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS * FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) =
M-NEXT-NODE-DETECTION

Reading FIRST

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST *
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) =
M-NEXT-NODE-DETECTION

Reading DETECTED

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED * BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) =
M-NEXT-NODE-DETECTION

Reading BY

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY * THE (DETECTION-COMPONENT M-ORU) *PERIOD*) =
M-NEXT-NODE-DETECTION

Reading THE

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY THE * (DETECTION-COMPONENT M-ORU) *PERIOD*) =
M-NEXT-NODE-DETECTION

Reading *QUOTE*

Reading NEXT

(*QUOTE* NEXT * *QUOTE* ACTIVE NODE ON THE NETWORK) = M-NEXT-RC
(NEXT * ACTIVE RING CONCENTRATOR IN THE NETWORK) = M-NEXT-RC

Just as with parentheses, quotes are not yet handled in a systematic, general manner.

Reading *QUOTE*

(*QUOTE* NEXT *QUOTE* * ACTIVE NODE ON THE NETWORK) = M-NEXT-RC

Reading ACTIVE

(*QUOTE* NEXT *QUOTE* ACTIVE * NODE ON THE NETWORK) = M-NEXT-RC

Reading NODE

(*QUOTE* NEXT *QUOTE* ACTIVE NODE * ON THE NETWORK) = M-NEXT-RC

Reading ON

(*QUOTE* NEXT *QUOTE* ACTIVE NODE ON * THE NETWORK) = M-NEXT-RC

Reading THE
(*QUOTE* NEXT *QUOTE* ACTIVE NODE ON THE * NETWORK) = M-NEXT-RC

Reading NETWORK
(DMS NETWORK *) = M-CORE-NETWORK referenced
Specializing: I-M-CORE-NETWORK.471
(*QUOTE* NEXT *QUOTE* ACTIVE NODE ON THE NETWORK *) = M-NEXT-RC referenced
Specializing: I-M-NEXT-RC.472
(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) * *PERIOD*) =
M-NEXT-NODE-DETECTION

Reading *PERIOD*
(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT I-M-NEXT-RC.472)
PERIOD *) = M-NEXT-NODE-DETECTION referenced
Specializing: I-M-NEXT-NODE-DETECTION.474
((PROCEDURE M-FAILURE-DETECTION) * (EVENT M-EVENT-STEP)) =
M-DISAMBIGUATE-DETECTION-EVENT
Specializing: I-M-HL-PATTERN.409.476
Specializing: I-M-HL-PATTERN.387.477
Specializing: I-M-HL-PATTERN.416.478
Removing: I-M-IP.407.457
Removing: I-M-IP.328.467
Removing: I-M-IP.395.458
Removing: I-M-IP.380.453
Removing: I-M-IP.371.428

Here a piece of the detection procedure from the text is recognized by activating a detection procedure called M-NEXT-NODE-DETECTION detailing the failed and detection components. Over the course of parsing the detection section, several such detection procedures may be instantiated as different parts of the procedure are recognized. All of these procedural fragments will ultimately be unified into one detection procedure when the context changes. The first sentence is in effect acting as a pointer to the type of procedure that is being described; the class of M-NEXT-NODE-DETECTION organizes procedures which involve one node noticing that another node has failed to communicate with it as expected, indicating a possible failure of that node. The first high level pattern I-M-HL-PATTERN.409.476 is the unification pattern which collects together instances of detection procedures for unification at a later time. The second pattern I-M-HL-PATTERN.387.477 is the one which gathers detection procedures together for inclusion in the case description. The last high level pattern is one which deals with building procedures from individual events in a procedure. The latter one is not relevant to continued processing. Notice that several index patterns are purged from the system at this time. Since the "*period*" was processed, the system assumes that the end of a sentence was encountered, and purges any useless patterns that were designed (perhaps by default) to be killed at sentence boundaries.

Reading SYSTEM

(SYSTEM *) = M-SYSTEMS referenced

Specializing: I-M-SYSTEMS.480

((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*

(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

(SYSTEM * MANAGEMENT) = M-SM

Reading MANAGEMENT

(SYSTEM MANAGEMENT *) = M-SM referenced

Recognizing: M-SM

Creating: I-M-SM.483

Removing: I-M-SM.483

Specializing: I-M-SYSTEM-SM

Here is an example where an activated MOP specialized into a MOP that is already resident in memory. In this case, I-M-SYSTEM-SM happened to be predefined within the system. The parser makes use of pre-existing memory structures whenever possible.

Reading WILL

Reading REACH

Reading A

(*LEFT-PAREN* A * *RIGHT-PAREN*) = M ENUMERATION

Reading TIME-OUT

((RECIPIENT I-M-SYSTEM-SM) WILL REACH A TIME-OUT * LIMIT FOR RECEIPT
OF THE (MESSAGE M-MESSAGE) *PERIOD*) = M-TIME-OUT-EVENT

Reading LIMIT

((RECIPIENT I-M-SYSTEM-SM) WILL REACH A TIME-OUT LIMIT * FOR RECEIPT
OF THE (MESSAGE M-MESSAGE) *PERIOD*) = M-TIME-OUT-EVENT

Reading FOR

((RECIPIENT I-M-SYSTEM-SM) WILL REACH A TIME-OUT LIMIT FOR * RECEIPT
OF THE (MESSAGE M-MESSAGE) *PERIOD*) = M-TIME-OUT-EVENT

Reading RECEIPT

((RECIPIENT I-M-SYSTEM-SM) WILL REACH A TIME-OUT LIMIT FOR RECEIPT *
OF THE (MESSAGE M-MESSAGE) *PERIOD*) = M-TIME-OUT-EVENT

Reading OF

((RECIPIENT I-M-SYSTEM-SM) WILL REACH A TIME-OUT LIMIT FOR RECEIPT
OF * THE (MESSAGE M-MESSAGE) *PERIOD*) = M-TIME-OUT-EVENT

Reading THE

((RECIPIENT I-M-SYSTEM-SM) WILL REACH A TIME-OUT LIMIT FOR RECEIPT
OF THE * (MESSAGE M-MESSAGE) *PERIOD*) = M-TIME-OUT-EVENT

Reading NETWORK

(DMS NETWORK *) = M-CORE-NETWORK referenced

Specializing: I-M-CORE-NETWORK.471

Although a pattern causes M-CORE-NETWORK to be recognized in the text (the "DMS" part of the phrase is optional), no useful processing occurs, since there were no expectations for that MOP.

Reading TOKEN

(NETWORK TOKEN *) = M-TOKEN referenced

Specializing: I-M-TOKEN.491

((RECIPIENT I-M-SYSTEM-SM) WILL REACH A TIME-OUT LIMIT FOR RECEIPT
OF THE (MESSAGE M-MESSAGE) * *PERIOD*) = M-TIME-OUT-EVENT

Reading *PERIOD*

((RECIPIENT I-M-SYSTEM-SM) WILL REACH A TIME-OUT LIMIT FOR RECEIPT
OF THE (MESSAGE I-M-TOKEN.491) *PERIOD* *) = M-TIME-OUT-EVENT
referenced

Specializing: I-M-TIME-OUT-EVENT.492

Specializing: I-M-HL-PATTERN.416.505

((PROCEDURE I-M-NEXT-NODE-DETECTION.474)

(EVENT I-M-TIME-OUT-EVENT.492 *) =

M-DISAMBIGUATE-DETECTION-EVENT referenced

Specializing: I-M-DISAMBIGUATE-DETECTION-EVENT.506

Creating: I-M-TIME-OUT-EVENT.144.507

Recognizing: I-M-TIME-OUT-EVENT.144.507

Here an event is recognized from the text. Events cause the activation of a high level pattern that collects together those events occurring within a context. When a context switch occurs, the pattern MOP will search for the procedure that best organizes all of the events, and will recognize that procedure as the detection procedure being described. The MOP M-DISAMBIGUATE-DETECTION-EVENT ties together the procedure recognized from the first sentence to this event, such that some of the slots for the event can be fleshed out. Detection procedures are typically described by a pointer to the procedural class followed by an event from the procedure. This index pattern captures that stereotypical pattern of text usage in the FMEA manuals, acting as a processing aid for disambiguating the actual event.

Reading CORRECTIVE

(CORRECTIVE * ACTION *COLON*) = M-CORRECTION-CONTEXT

Reading ACTION

(CORRECTIVE ACTION * *COLON*) = M-CORRECTION-CONTEXT

Reading *COLON*

(CORRECTIVE ACTION *COLON* *) = M-CORRECTION-CONTEXT referenced

Specializing: I-M-CORRECTION-CONTEXT.531

Specializing: I-M-CONTEXT.603

Specializing: I-M-BUILD-PROCEDURE.533

Specializing: I-M-AND-GROUP.145.534

Specializing: I-M-BUILD-PROCEDURE.557

Specializing: I-M-DETECTION-PROC.146.558

Specializing: I-M-BUILD-PROCEDURE.560

Specializing: I-M-DETECTION-AND-GROUP.147.561

Specializing: I-M-BUILD-PROCEDURE.588

Specializing: I-M-NEXT-NODE-DETECTION.589

((PROCEDURE M-FAILURE-DETECTION) * (EVENT M-EVENT-STEP)) =

```
M-DISAMBIGUATE-DETECTION-EVENT
Specializing: I-M-UNIFY-DETECTION-MOPS.595
Specializing: I-M-NEXT-NODE-DETECTION.596
((PROCEDURE M-FAILURE-DETECTION) * (EVENT M-EVENT-STEP)) =
    M-DISAMBIGUATE-DETECTION-EVENT
```

As usual, with the recognition of a new context, the high level processing MOPs of the prior context are resolved and terminated. In this case, there are several high level patterns that were operating in the old context. The first was a pattern that collected together any referenced events (there was only one in this particular detection description). This pattern activates the M-BUILD-PROCEDURE MOP, which searches through memory for procedures which organize those events. Procedures are typically built up recursively; in this example, the I-M-NEXT-NODE-DETECTION.589 procedure is built up over several steps. It is important to note that this process is one of recognition. If there is no procedure in memory to organize the given events, a detection procedure cannot be recognized for the text. Once the events have been handled, another high level pattern is resolved. This one collects together procedures that occur in a context and unifies them (if they are of the same class) into a single procedure, collecting together all of the slots which characterize that MOP. In this case, the procedure created by the first sentence and the procedure built up from the event are unified together into a single detection procedure called I-M-NEXT-NODE-DETECTION.596. After this point, the detection procedure is added to the case description, and the parsing continues.

Reading *LEFT-PAREN*

Reading A
(*LEFT-PAREN* A * *RIGHT-PAREN*) = M ENUMERATION

Reading *RIGHT-PAREN*

(*LEFT-PAREN* A *RIGHT-PAREN* *) = M ENUMERATION referenced
Specializing: I-M ENUMERATION.605
Specializing: I-M BUILD CONTEXT.606
Specializing: I-M CONTEXT.608

An enumerated list is processed by the parser by creating new subcontexts each time an enumeration tag is specified. The parser currently recognizes the tags of (a), (b), ..., as an enumerated sequence. The strategy to handle this is simple: a context switch is made to wrap up any of the higher level patterns for the prior context. Then that context is reasserted. Thus if an enumerated list with two enumerations occurs in the correction procedures section, the correction context will be asserted twice. This has the effect of wrapping up any processing that needs to be resolved, while still maintaining a given context.

Reading SHORT
(SHORT * TERM) = M SHORT TERM

Reading TERM
(SHORT TERM *) = M SHORT TERM referenced
Specializing: I-M SHORT TERM.610
((FRAMEWORK M FRAMEWORK) * *COLON*) = M FAILURE CORRECTION

Each correction procedure has a framework specifier of either short-term or long-term.

Reading *COLON*

((FRAMEWORK I-M-SHORT-TERM.610) *COLON* *) = M-FAILURE-CORRECTION referenced
Specializing: I-M-FAILURE-CORRECTION.612
Specializing: I-M-HL-PATTERN.404.613
Specializing: I-M-HL-PATTERN.387.614
Specializing: I-M-HL-PATTERN.416.615

A failure correction MOP is instantiated here with the framework recorded as a slot filler. Just as with the detection procedure, multiple correction procedures may be activated, each of which records some information about the correction procedure being described in the text. These will be unified at the next context switch into the final correction procedure. The high level patterns are those that are concerned with building up procedures (which, recall, may be composed of events and other procedures), with unification, and with building the case.

Reading NETWORK

(DMS NETWORK *) = M-CORE-NETWORK referenced
Specializing: I-M-CORE-NETWORK.471

Reading RECONFIGURATION

((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION * IS EFFECTED
AUTOMATICALLY *PERIOD*) = M-AUTO-RECONFIG

Reading IS

((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION IS * EFFECTED
AUTOMATICALLY *PERIOD*) = M-AUTO-RECONFIG

Reading EFFECTED

((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION IS EFFECTED *
AUTOMATICALLY *PERIOD*) = M-AUTO-RECONFIG

Reading AUTOMATICALLY

((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION IS EFFECTED
AUTOMATICALLY *PERIOD*) = M-AUTO-RECONFIG

Reading *PERIOD*

((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION IS EFFECTED
AUTOMATICALLY *PERIOD*) = M-AUTO-RECONFIG referenced

Specializing: I-M-AUTO-RECONFIG.648

Inferring--> M-RECONFIGURATION

Specializing: I-M-RECONFIGURATION.649

Specializing: I-M-HL-PATTERN.404.650

Specializing: I-M-HL-PATTERN.387.651

The MOP M-AUTO-RECONFIG characterizes the process of a network reconfiguring itself under software control. The major part of this act is described by the M-RECONFIGURATION class of correction procedures, which organizes those procedures that reconfigure the network in some manner; the functionality of the M-AUTO-RECONFIG MOP therefore infers the M-RECONFIGURATION procedure. Since this is a type of correction procedure, it will eventually be unified along with the other correction procedure fragments.

Reading THE

Reading DMS

Reading NETWORK

(DMS NETWORK *) = M-CORE-NETWORK referenced
Specializing: I-M-CORE-NETWORK.471

Reading REMAINS

(THE (OBJECT I-M-CORE-NETWORK.471) REMAINS * IN OPERATION) = M-OPERATIONAL

Reading IN

(THE (OBJECT I-M-CORE-NETWORK.471) REMAINS IN * OPERATION) = M-OPERATIONAL

Reading OPERATION

(THE (OBJECT I-M-CORE-NETWORK.471) REMAINS IN OPERATION *) =
M-OPERATIONAL referenced
Specializing: I-M-OPERATIONAL.670

This is the first example of a state that has been explicitly mentioned in the text. Other states have already been implicitly instantiated, however, as preconditions and results to the events and procedures that have been recognized previously in the text.

Reading IN

Reading A

(*LEFT-PAREN* A * *RIGHT-PAREN*) = M ENUMERATION

Reading RECONFIGURED

((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED * STATE WITH THE
FAILED (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE

Reading STATE

((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED STATE * WITH THE
FAILED (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE

Reading WITH

((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED STATE WITH * THE
FAILED (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE

Reading THE

((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED STATE WITH THE *
FAILED (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE

Reading FAILED

((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED STATE WITH THE
FAILED * (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE

Reading RC

(RC *) = M-RC referenced
Specializing: I-M-RC.460
((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED STATE WITH THE
FAILED (FAILED-COMPONENT M-ORU) * BYPASSED) = M-BYPASS-NODE

Reading BYPASSED

((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED STATE WITH THE

FAILED (FAILED-COMPONENT I-M-RC.460) BYPASSED *) =

M-BYPASS-NODE referenced

Specializing: I-M-BYPASS-NODE.681

Specializing: I-M-HL-PATTERN.404.682

Specializing: I-M-HL-PATTERN.387.683

Specializing: I-M-HL-PATTERN.416.684

Here a more specific instance of the M-RECONFIGURATION class has been activated.
The high level patterns are the usual ones activated with a procedure.

Reading *PERIOD*

Removing: I-M-IP.395.672

Reading *LEFT-PAREN*

Reading B

(*LEFT-PAREN* B * *RIGHT-PAREN*) = M-ENUMERATION

Reading *RIGHT-PAREN*

(*LEFT-PAREN* B *RIGHT-PAREN* *) = M-ENUMERATION referenced

Specializing: I-M-ENUMERATION.605

Specializing: I-M-BUILD-CONTEXT.606

Specializing: I-M-CONTEXT.697

Removing: I-M-HL-PATTERN.416.684

Removing: I-M-HL-PATTERN.416.652

Removing: I-M-HL-PATTERN.416.615

Specializing: I-M-BUILD-PROCEDURE.689

Removing: I-M-HL-PATTERN.404.682

Removing: I-M-HL-PATTERN.404.650

Removing: I-M-HL-PATTERN.404.613

Specializing: I-M-UNIFY-CORRECTION-MOPS.690

Specializing: I-M-BYPASS-NODE.691

Specializing: I-M-UNIFY-CORRECTION-MOPS.693

Specializing: I-M-HL-PATTERN.387.694

Specializing: I-M-BUILD-PROCEDURE.696

With the enumeration tag, we have a context switch again. The M-BUILD-PROCEDURE MOP attempts to find a procedure in memory which organizes any of the correction procedures that have been activated, since procedures may be composed of subprocedures. This task fails in this instance, since there are no such procedures involving these correction MOPs. The unification then occurs, creating the final instance of I-M-BYPASS-NODE.691. At this point, the correction procedure is added to the case description.

Reading LONG

(LONG * TERM) = M-LONG-TERM

Reading TERM

(LONG TERM *) = M-LONG-TERM referenced

Specializing: I-M-LONG-TERM.699

((FRAMEWORK M-FRAMEWORK) * *COLON*) = M-FAILURE-CORRECTION

Reading *COLON*

((FRAMEWORK I-M-LONG-TERM.699) *COLON* *) = M-FAILURE-CORRECTION referenced
Specializing: I-M-FAILURE-CORRECTION.701
Specializing: I-M-HL-PATTERN.404.702
Specializing: I-M-HL-PATTERN.387.703
Specializing: I-M-HL-PATTERN.416.704

Reading THE

Reading CREWMEN

(THE CREWMEN *) = M-CREW referenced
Specializing: I-M-CREW.737
((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading CHECK

((ACTOR I-M-CREW.737) CHECK * FOR (OBJECT M-SYSTEM)) = M-VERIFY-EVENT

Reading FOR

((ACTOR I-M-CREW.737) CHECK FOR * (OBJECT M-SYSTEM)) = M-VERIFY-EVENT

Reading *QUOTE*

Reading APPLIED

(*QUOTE* APPLIED * POWER *QUOTE*) = M-POWER-SYSTEM

Reading POWER

(*QUOTE* APPLIED POWER * *QUOTE*) = M-POWER-SYSTEM

Reading *QUOTE*

(*QUOTE* APPLIED POWER *QUOTE* *) = M-POWER-SYSTEM referenced
Specializing: I-M-POWER-SYSTEM.308
((ACTOR I-M-CREW.737) CHECK FOR (OBJECT I-M-POWER-SYSTEM.308) *) =
M-VERIFY-EVENT referenced
Specializing: I-M-VERIFY-EVENT.742
Specializing: I-M-HL-PATTERN.416.745

Note that although the event describing the verification of the power system has been activated, it is not clear what the power system is a component of from this sentence. A context for this event will eventually be provided when the correction procedure which organizes it is recognized. At that time, the device under consideration (a ring concentrator) will be referenced along with this event, thereby providing the explicit connection between the power system and the ring concentrator.

Reading AND

Reading THE

Reading CREWMEN

(THE CREWMEN *) = M-CREW referenced
Specializing: I-M-CREW.737
((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading CHECK
((ACTOR I-M-CREW.737) CHECK * FOR (OBJECT M-SYSTEM)) = M-VERIFY-EVENT

Reading FOR
((ACTOR I-M-CREW.737) CHECK FOR * (OBJECT M-SYSTEM)) = M-VERIFY-EVENT

Reading *QUOTE*

Reading CONNECTOR
(*QUOTE* CONNECTOR * TIGHTNESS *QUOTE*) = M-CONNECTORS

Reading TIGHTNESS
(*QUOTE* CONNECTOR TIGHTNESS * *QUOTE*) = M-CONNECTORS

Reading *QUOTE*
(*QUOTE* CONNECTOR TIGHTNESS *QUOTE* *) = M-CONNECTORS referenced
Specializing: I-M-CONNECTORS.310
((ACTOR I-M-CREW.737) CHECK FOR (OBJECT I-M-CONNECTORS.310) *) =
M-VERIFY-EVENT referenced
Specializing: I-M-VERIFY-EVENT.754
Specializing: I-M-HL-PATTERN.416.757

Here is a second event from the current correction procedure. This illustrates how multiple events from a procedure may be specified within a context.

Reading *PERIOD*

Reading IF

Reading THE

Reading RC
(RC *) = M-RC referenced
Specializing: I-M-RC.460

Reading CANNOT

Reading THEN

Reading BE

Reading PLACED

Reading IN

Reading OPERATION

Reading *COMMA*

Reading THE

Reading RC

(RC *) = M-RC referenced

Specializing: I-M-RC.460

Reading IS

Reading REMOVED

Reading AND

Reading REPLACED

(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED * WITH
AN ORU LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading WITH

(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH *
AN ORU LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading AN

(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
* ORU LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading ORU

(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
ORU * LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading LOGISTICS

(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
ORU LOGISTICS * SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading SPARE

(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
ORU LOGISTICS SPARE * *PERIOD*) = M-REPLACE-WITH-SPARE

Reading *PERIOD*

(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
ORU LOGISTICS SPARE *PERIOD* *) = M-REPLACE-WITH-SPARE referenced

Specializing: I-M-REPLACE-WITH-SPARE.778

Specializing: I-M-HL-PATTERN.404.779

Specializing: I-M-HL-PATTERN.387.780

Specializing: I-M-HL-PATTERN.416.781

Note that this sentence begins with an "if" conditional clause. This is ignored by the current system, since conditionals are not represented within the knowledge representation. This decision was motivated by the desire to simplify some of the issues involved in parsing text; the addition of conditionals would have significantly complicated processing.

Reading FAILURE

Reading EFFECT

(FAILURE EFFECT * ON *COLON*) = M-EFFECT-CONTEXT

(FAILURE EFFECT * ON THE (EFFECTED-COMPONENT M-MISSION-COMPONENT)
COLON (SEQUENCE-OF-STEPS M-GROUP)) = M-FAILURE-EFFECT

Reading ON

(FAILURE EFFECT ON * THE (EFFECTED-COMPONENT M-MISSION-COMPONENT)
COLON (SEQUENCE-OF-STEPS M-GROUP)) = M-FAILURE-EFFECT
(FAILURE EFFECT ON * "COLON") = M-EFFECT-CONTEXT

Reading *COLON*

(FAILURE EFFECT ON *COLON* *) = M-EFFECT-CONTEXT referenced
Specializing: I-M-EFFECT-CONTEXT.784
Specializing: I-M-BUILD-PROCEDURE.786
Specializing: I-M-AND-GROUP.313.787
Specializing: I-M-BUILD-PROCEDURE.826
Specializing: I-M-AND-GROUP.314.827
Specializing: I-M-BUILD-PROCEDURE.885
Specializing: I-M-CORRECTION-PROC.315.886
Specializing: I-M-BUILD-PROCEDURE.888
Specializing: I-M-CORRECTION-AND-GROUP.316.889
Specializing: I-M-BUILD-PROCEDURE.931
Specializing: I-M-NON-OPERATIONAL-REPLACEMENT.932
Specializing: I-M-HL-PATTERN.404.933
Specializing: I-M-HL-PATTERN.387.934
Specializing: I-M-BUILD-PROCEDURE.936
Specializing: I-M-UNIFY-CORRECTION-MOPS.937
Specializing: I-M-FAILURE-CORRECTION.938
Specializing: I-M-CONTEXT.956

Once again, the build procedure and unification processes kick in and create the correction procedure I-M-NON-OPERATIONAL-REPLACEMENT.932 described in the text. In this instance, the parser has to recognize a correction procedure that organizes three events mentioned in the text. The M-BUILD-PROCEDURE is general enough to find a candidate procedure that best characterizes any number of events. At this point in the parsing of the text, both of the correction procedures have been added to the case description.

Reading *LEFT-PAREN*

Reading A

(*LEFT-PAREN* A * "RIGHT-PAREN") = M-ENUMERATION

Reading *RIGHT-PAREN*

(*LEFT-PAREN* A *RIGHT-PAREN* *) = M-ENUMERATION referenced
Specializing: I-M-ENUMERATION.605
Specializing: I-M-BUILD-CONTEXT.959
Specializing: I-M-CONTEXT.961

Reading CREW

(CREW *) = M-CREW referenced
Specializing: I-M-CREW.737
((EFFECTED-COMPONENT M-MISSION-COMPONENT) * "COLON"
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading *SLASH*

Reading SSPE

(CREW *SLASH* SSPE *) = M-CREW referenced
Specializing: I-M-CREW.737
((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading *COLON*

((EFFECTED-COMPONENT I-M-CREW.737) *COLON* *
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT
((EFFECTED-COMPONENT I-M-CREW.737) *COLON* *
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading NONE

(NONE * *PERIOD*) = M-DISAMBIGUATE-NONE

Reading *PERIOD*

(NONE *PERIOD* *) = M-DISAMBIGUATE-NONE referenced
Specializing: I-M-DISAMBIGUATE-NONE.969
Recognizing: I-M-NULL-PROCEDURE
((EFFECTED-COMPONENT I-M-CREW.737) *COLON*
(SEQUENCE-OF-STEPS I-M-NULL-PROCEDURE *) = M-FAILURE-EFFECT
referenced
Specializing: I-M-FAILURE-EFFECT.970
((SEQUENCE-OF-STEPS M-FAILURE-EFFECT)*
(JUSTIFICATION M-FAILURE-CORRECTION)) = M-SEARCH-FOR-EFFECT-REFERENT
Specializing: I-M-HL-PATTERN.387.972
Specializing: I-M-HL-PATTERN.416.973

The word "none" when used in the context of a failure effect usually indicates that there is no effect. That same word in another context might mean something entirely different. The MOP M-DISAMBIGUATE-NONE checks for the current context to disambiguate the meaning of that word. There are certainly other (possibly more general) methods of disambiguating a word within a given context; this one was chosen for ease of implementation. "None" in this context therefore causes the activation of I-M-NULL-PROCEDURE, the concept that is used to represent the lack of a procedure.

Reading THE

Reading CONDITIONS

(THE CONDITIONS * ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN
THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES
PERIOD) = M-BYPASS-NODE

This pattern demonstrates that index patterns can be quite specific and therefore not of general utility if not defined properly. This pattern is effective only for a sentence that follows this precise format. A better definition of the pattern would be to generalize it. Minimally, each of the entities mentioned could be replaced by an ORU concept filler, which would allow the phrase to be used in conjunction with any of the ORU devices. Since this is such a long phrase, it will be omitted from the trace until it is actually recognized.

Reading ASSOCIATED

Reading WITH

Reading THE

Reading REPLACEMENT

Reading OF

Reading A

(*LEFT-PAREN* A * *RIGHT-PAREN*) = M-ENUMERATION

Reading RC

(RC *) = M-RC referenced

Specializing: I-M-RC.460

Reading WITHIN

Reading THE

Reading NETWORK

(DMS NETWORK *) = M-CORE-NETWORK referenced

Specializing: I-M-CORE-NETWORK.471

Reading CONFIGURATION

Reading ARE

Reading SUCH

Reading THAT

Reading THE

Reading NETWORK

(DMS NETWORK *) = M-CORE-NETWORK referenced

Specializing: I-M-CORE-NETWORK.471

Reading IS

Reading ALREADY

Reading IN

Reading A

(*LEFT-PAREN* A * *RIGHT-PAREN*) = M-ENUMERATION

Reading RECONFIGURED

((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED * STATE WITH THE
FAILED (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE

Reading STATE

((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED STATE * WITH THE
FAILED (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE

Reading SUPPLYING

Reading FULL

Reading SERVICES

Reading *PERIOD*

(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD* *) =
M-BYPASS-NODE referenced

Specializing: I-M-BYPASS-NODE.996

Specializing: I-M-HL-PATTERN.404.997

((SEQUENCE-OF-STEPS I-M-FAILURE-EFFECT.970)

(JUSTIFICATION I-M-BYPASS-NODE.996) *) =

M-SEARCH-FOR-EFFECT-REFERENT referenced

Specializing: I-M-SEARCH-FOR-EFFECT-REFERENT.999

Specializing: I-M-HL-PATTERN.416.1000

With failure effects, the FMEA manuals generally follow one of two patterns. They either detail a procedure that represents the effect on a system component (this may be the null procedure), or they specify a procedure and give a justification for those effects which references one of the correction procedures in the same case. The latter instance occurs with this first failure effect. The justification detailed is the M-BYPASS-NODE procedure from the failure corrections section (the argument is essentially that there is no effect because the failed component is automatically bypassed). When such a justification is given, the parser must search back over the failure correction procedures to determine which one is being used as the justification. The M-BYPASS-NODE procedure instantiated by the justification is used to find the matching M-BYPASS-PROCEDURE from the corrections section; in this case, I-M-BYPASS-NODE.691 is found (although this is not reflected in the trace). This procedure is then used as the filler for the failure effect slot.

Reading *LEFT-PAREN*

Reading B

(*LEFT-PAREN* B *RIGHT-PAREN*) = M-ENUMERATION

Reading *RIGHT-PAREN*

(*LEFT-PAREN* B *RIGHT-PAREN* *) = M-ENUMERATION referenced

Specializing: I-M-ENUMERATION.605

Specializing: I-M-BUILD-CONTEXT.959

Specializing: I-M-CONTEXT.1007

With the new failure effect context being established, the first failure effect is added to the case description.

Reading MISSION

(MISSION * SUPPORT) = M-MISSION-SUPPORT

Reading SUPPORT

(MISSION SUPPORT *) = M-MISSION-SUPPORT referenced

Specializing: I-M-MISSION-SUPPORT.1009

((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading *COLON*

((EFFECTED-COMPONENT I-M-MISSION-SUPPORT.1009) *COLON* *
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading NONE

(NONE * *PERIOD*) = M-DISAMBIGUATE-NONE

Reading *PERIOD*

(NONE *PERIOD* *) = M-DISAMBIGUATE-NONE referenced

Specializing: I-M-DISAMBIGUATE-NONE.969

Recognizing: I-M-NUL-PROCEDURE

((EFFECTED-COMPONENT I-M-MISSION-SUPPORT.1009) *COLON*
(SEQUENCE-OF-STEPS I-M-NULL-PROCEDURE) *) = M-FAILURE-EFFECT
referenced
Specializing: I-M-FAILURE-EFFECT.1013
((SEQUENCE-OF-STEPS M-FAILURE-EFFECT) *
(JUSTIFICATION M-FAILURE-CORRECTION)) = M-SEARCH-FOR-EFFECT-REFERENT
Specializing: I-M-HL-PATTERN.387.1015
Specializing: I-M-HL-PATTERN.416.1016

Reading *LEFT-PAREN*

Reading C

(*LEFT-PAREN* C * *RIGHT-PAREN*) = M-ENUMERATION

Reading *RIGHT-PAREN*

(*LEFT-PAREN* C *RIGHT-PAREN* *) = M-ENUMERATION referenced

Specializing: I-M-ENUMERATION.605

Specializing: I-M-BUILD-CONTEXT.959

Specializing: I-M-CONTEXT.1023

The second failure effect is now added to the case description. This one has just the null procedure as its filler.

Reading SYSTEM

(SYSTEM *) = M-SYSTEMS referenced

Specializing: I-M-SYSTEMS.480

((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT
(SYSTEM * MANAGEMENT) = M-SM

Reading *COLON*

((EFFECTED-COMPONENT I-M-SYSTEMS.480) *COLON* *
(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading NONE

(NONE * *PERIOD*) = M-DISAMBIGUATE-NONE

Reading *PERIOD*

(NONE *PERIOD* *) = M-DISAMBIGUATE-NONE referenced

Specializing: I-M-DISAMBIGUATE-NONE.969

Recognizing: I-M-NULL-PROCEDURE

((EFFECTED-COMPONENT I-M-SYSTEMS.480) *COLON*

(SEQUENCE-OF-STEPs I-M-NULL-PROCEDURE) *) = M-FAILURE-EFFECT

referenced

Specializing: I-M-FAILURE-EFFECT.1030

((SEQUENCE-OF-STEPs M-FAILURE-EFFECT) *

(JUSTIFICATION M-FAILURE-CORRECTION)) = M-SEARCH-FOR-EFFECT-REFERENT

Specializing: I-M-HL-PATTERN.387.1032

Specializing: I-M-HL-PATTERN.416.1033

Reading *LEFT-PAREN*

Reading D

(*LEFT-PAREN* D * *RIGHT-PAREN*) = M-ENUMERATION

Reading *RIGHT-PAREN*

(*LEFT-PAREN* D *RIGHT-PAREN* *) = M-ENUMERATION referenced

Specializing: I-M-ENUMERATION.605

Recognizing: M-CONTEXT

Specializing: I-M-CONTEXT.1040

The third failure effect is added to the case description at this point.

Reading INTERFACES

(INTERFACES *) = M-INTERFACES referenced

Specializing: I-M-INTERFACES.1042

((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*

(SEQUENCE-OF-STEPs M-PROCEDURE)) = M-FAILURE-EFFECT

Reading *COLON*

((EFFECTED-COMPONENT I-M-INTERFACES.1042) *COLON* *)

(SEQUENCE-OF-STEPs M-PROCEDURE)) = M-FAILURE-EFFECT

Reading NONE

(NONE * *PERIOD*) = M-DISAMBIGUATE-NONE

Reading *PERIOD*

(NONE *PERIOD* *) = M-DISAMBIGUATE-NONE referenced

Specializing: I-M-DISAMBIGUATE-NONE.969

Recognizing: I-M-NULL-PROCEDURE

((EFFECTED-COMPONENT I-M-INTERFACES.1042) *COLON*

(SEQUENCE-OF-STEPs I-M-NULL-PROCEDURE) *) = M-FAILURE-EFFECT

referenced

Specializing: I-M-FAILURE-EFFECT.1046

((SEQUENCE-OF-STEPs M-FAILURE-EFFECT) *)

(JUSTIFICATION M-FAILURE-CORRECTION)) = M-SEARCH-FOR-EFFECT-REFERENT

Specializing: I-M-HL-PATTERN.387.1048

Specializing: I-M-HL-PATTERN.416.1049

Reading *EOC*

Specializing: I-M-CONTEXT.1051

Specializing: I-M-CASE.1053

The *EOC* is used to mark the end of a case text. It also causes a context switch, which wraps up the processing of the last effect. Finally, the *EOC* symbol causes the termination of the high level pattern that builds up case descriptions. The final case structure is now instantiated, and the parsing process is complete for the given input.

This is the final case description that is built by the parser:

```
>(dph `I-M-CASE.1053)
(I-M-CASE.1053 (FAILED-COMPONENT I-M-GENERIC-RC.422)
  (MODE I-M-STARTUP-NO-OUTPUT.1089
    (SEQUENCE-OF-STEPS I-M-AND-GROUP.97.1085
      (2 I-M-WAIT-TO-RECEIVE-EVENT.95.1071
        (RESULT I-M-TIME-ELAPSED-FOR-OBJECT.25.1056
          (OBJECT I-M-SYSTEM-SM))
        (RESULT I-M-READY-TO-RECEIVE.22.1059
          (RECIPIENT I-M-SYSTEM-SM)
          (OBJECT I-M-TOKEN.491))
        (ACTION I-M-WAIT-FOR-MESSAGE.23.1064
          (ACTOR I-M-SYSTEM-SM) (RECIPIENT I-M-SYSTEM-SM)
          (OBJECT I-M-TOKEN.491))
        (PRECOND I-M-READY-TO-RECEIVE.22.1059
          (RECIPIENT I-M-SYSTEM-SM)
          (OBJECT I-M-TOKEN.491))
        (RECIPIENT I-M-SYSTEM-SM) (MESSAGE I-M-TOKEN.491)))
      (3 I-M-TIME-OUT-EVENT.96.1084
        (RESULT I-M-TRANSMISSION-TIMED-OUT.41.502
          (RECIPIENT I-M-SYSTEM-SM)
          (OBJECT I-M-TOKEN.491))
        (ACTION I-M-TIME-OUT.40.499 (ACTOR I-M-SYSTEM-SM)
          (RECIPIENT I-M-SYSTEM-SM)
          (OBJECT I-M-TOKEN.491))
        (PRECOND I-M-READY-TO-RECEIVE.36.495
          (RECIPIENT I-M-SYSTEM-SM)
          (OBJECT I-M-TOKEN.491))
        (RECIPIENT I-M-SYSTEM-SM) (MESSAGE I-M-TOKEN.491)))
      (PRECOND I-M-READY-TO-RECEIVE.93.1088
        (RECIPIENT I-M-SYSTEM-SM) (OBJECT I-M-TOKEN.491)))
    (CAUSE I-M-FAULTY-COMPONENT-CAUSE.1094
    (CAUSE I-M-CONTAMINATION-CAUSE.440)
    (CAUSE I-M-TEMPERATURE-OUT-OF-RANGE-CAUSE.443)
    (CAUSE I-M-MECHANICAL-SHOCK-CAUSE.446)
    (CAUSE I-M-THERMAL-SHOCK-CAUSE.449)
    (DETECTION I-M-NEXT-NODE-DETECTION.1150
      (RESULT I-M-NON-OPERATIONAL.148.1100
        (OBJECT I-M-GENERIC-RC.422))
      (PRECOND I-M-POWER-ON.141.1102 (OBJECT I-M-NEXT-RC.472))
      (PRECOND I-M-NEXT-NODE.140.1105 (OBJECT1 I-M-NEXT-RC.472)
        (OBJECT2 I-M-GENERIC-RC.422))
      (PRECOND I-M-LOGICALLY-CONNECTED.137.1107
```

(OBJECT1 I-M-GENERIC-RC.422))
 (PRECOND I-M-PHYSICALLY-CONNECTED.136.1109
 (OBJECT1 I-M-GENERIC-RC.422))
 (FAILED-COMPONENT I-M-GENERIC-RC.422)
 (DETECTION-COMPONENT I-M-NEXT-RC.472)
 (SEQUENCE-OF-STEPS I-M-DETECTION-AND-GROUP.147.1147
 (1 I-M-DETECTION-PROC.146.1144
 (PRECOND I-M-READY-TO-RECEIVE.142.563
 (OBJECT I-M-TOKEN.491))
 (SEQUENCE-OF-STEPS I-M-AND-GROUP.145.1142
 (1 I-M-WAIT-TO-RECEIVE-EVENT.143.1123
 (RESULT I-M-TIME-ELAPSED-FOR-OBJECT.25.1056
 (OBJECT I-M-SYSTEM-SM))
 (RESULT I-M-READY-TO-RECEIVE.22.536
 (OBJECT I-M-TOKEN.491))
 (ACTION I-M-WAIT-FOR-MESSAGE.23.538
 (OBJECT I-M-TOKEN.491))
 (PRECOND I-M-READY-TO-RECEIVE.22.536
 (OBJECT I-M-TOKEN.491))
 (MESSAGE I-M-TOKEN.491))
 (2 I-M-TIME-OUT-EVENT.144.507
 (PRECOND I-M-READY-TO-RECEIVE.36.495
 (RECIPIENT I-M-SYSTEM-SM)
 (OBJECT I-M-TOKEN.491))
 (ACTION I-M-TIME-OUT.40.499 (ACTOR I-M-SYSTEM-SM)
 (RECIPIENT I-M-SYSTEM-SM)
 (OBJECT I-M-TOKEN.491))
 (RESULT I-M-TRANSMISSION-TIMED-OUT.41.502
 (RECIPIENT I-M-SYSTEM-SM)
 (OBJECT I-M-TOKEN.491))
 (RECIPIENT I-M-SYSTEM-SM) (MESSAGE I-M-TOKEN.491))))))
 (CORRECTION I-M-BYPASS-NODE.1177
 (SEQUENCE-OF-STEPS I-M-CORRECTION-AND-GROUP.251.1164
 (1 I-M-CORRECTION-PROC.250.1163
 (SEQUENCE-OF-STEPS I-M-AND-GROUP.249.1162
 (1 I-M-LOGICAL-DISCONNECT-EVENT.248.1161
 (ACTOR I-M-SYSTEM-SM) (OBJECT1 I-M-GENERIC-RC.422)
 (OBJECT2 I-M-CORE-NETWORK.471)
 (PRECOND I-M-LOGICALLY-CONNECTED.245.1155
 (OBJECT2 I-M-CORE-NETWORK.471))
 (ACTION I-M-LOGICAL-DISCONNECT.246.1158
 (ACTOR I-M-SYSTEM-SM)
 (OBJECT2 I-M-CORE-NETWORK.471))
 (RESULT I-M-LOGICALLY-DISCONNECTED.247.1160
 (OBJECT2 I-M-CORE-NETWORK.471))))))
 (PRECOND I-M-PHYSICALLY-CONNECTED.202.1167
 (OBJECT1 I-M-GENERIC-RC.422)
 (OBJECT2 I-M-CORE-NETWORK.471))
 (PRECOND I-M-LOGICALLY-CONNECTED.203.1170
 (OBJECT1 I-M-GENERIC-RC.422)
 (OBJECT2 I-M-CORE-NETWORK.471))
 (FRAMEWORK I-M-SHORT-TERM.610)
 (CONFIGURATION I-M-CORE-NETWORK.471)
 (CORRECTION-COMPONENT I-M-SYSTEM-SM)
 (RESULT I-M-OPERATIONAL.302.1175 (OBJECT I-M-CORE-NETWORK.471)))

(FAILED-COMPONENT I-M-GENERIC-RC.422))
 (CORRECTION I-M-UNDEFINED-CORRECTION.1238
 (FRAMEWORK I-M-LONG-TERM.699)
 (FAILED-COMPONENT I-M-GENERIC-RC.422)
 (SEQUENCE-OF-STEPS I-M-CORRECTION-AND-GROUP.316.1237
 (1 I-M-CORRECTION-PROC.315.1236
 (SEQUENCE-OF-STEPS I-M-AND-GROUP.314.1235
 (1 I-M-AND-GROUP.313.1234
 (2 I-M-VERIFY-CONNECTORS-EVENT.311.793
 (ACTION I-M-VERIFY-CONNECTORS.84.790
 (ACTOR I-M-CREW.737)
 (OBJECT I-M-CONNECTORS.310))
 (ACTOR I-M-CREW.737) (OBJECT I-M-CONNECTORS.310))
 (1 I-M-VERIFY-POWER-EVENT.309.802
 (ACTION I-M-VERIFY-POWER.81.799 (ACTOR I-M-CREW.737)
 (OBJECT I-M-POWER-SYSTEM.308))
 (ACTOR I-M-CREW.737) (OBJECT I-M-POWER-SYSTEM.308))
 (3 I-M-REPLACE-WITH-SPARE.312.1229
 (RESULT I-M-LOGICALLY-CONNECTED.300.1193
 (OBJECT1 I-M-CORE-NETWORK.471))
 (RESULT I-M-OPERATIONAL.302.1175
 (OBJECT I-M-CORE-NETWORK.471))
 (PRECOND I-M-NON-OPERATIONAL.286.807
 (OBJECT I-M-GENERIC-RC.422))
 (SEQUENCE-OF-STEPS I-M-CORRECTION-AND-GROUP.299.1226
 (1 I-M-CORRECTION-PROC.298.1225
 (SEQUENCE-OF-STEPS I-M-AND-GROUP.297.1224
 (4 I-M-LOGICAL-CONNECT-EVENT.296.1199
 (OBJECT1 I-M-CORE-NETWORK.471))
 (2 I-M-PHYSICAL-CONNECT-EVENT.288.1207
 (RESULT I-M-PHYSICALLY-CONNECTED.73.1201
 (OBJECT1 I-M-CORE-NETWORK.471))
 (ACTION I-M-PHYSICAL-CONNECT.72.1203
 (OBJECT1 I-M-CORE-NETWORK.471))
 (PRECOND I-M-PHYSICALLY-DISCONNECTED.71.1205
 (OBJECT1 I-M-CORE-NETWORK.471))
 (OBJECT1 I-M-CORE-NETWORK.471))
 (1 I-M-PHYSICAL-DISCONNECT-EVENT.287.815
 (RESULT I-M-PHYSICALLY-DISCONNECTED.67.809
 (OBJECT1 I-M-GENERIC-RC.422))
 (ACTION I-M-PHYSICAL-DISCONNECT.66.811
 (OBJECT1 I-M-GENERIC-RC.422))
 (PRECOND I-M-PHYSICALLY-CONNECTED.65.813
 (OBJECT1 I-M-GENERIC-RC.422))
 (OBJECT1 I-M-GENERIC-RC.422))
 (3 I-M-LOGICAL-DISCONNECT-EVENT.292.857
 (RESULT I-M-LOGICALLY-DISCONNECTED.291.851
 (OBJECT1 I-M-GENERIC-RC.422))
 (ACTION I-M-LOGICAL-DISCONNECT.290.853
 (OBJECT1 I-M-GENERIC-RC.422))
 (PRECOND I-M-LOGICALLY-CONNECTED.289.855
 (OBJECT1 I-M-GENERIC-RC.422))
 (OBJECT1 I-M-GENERIC-RC.422))))
 (RESULT I-M-LOGICALLY-DISCONNECTED.301.822
 (OBJECT1 I-M-GENERIC-RC.422)))))))

(EFFECT I-M-FAILURE-EFFECT.1267
(SEQUENCE-OF-STEPS I-M-BYPASS-NODE.1177
(SEQUENCE-OF-STEPS I-M-CORRECTION-AND-GROUP.251.1164
(1 I-M-CORRECTION-PROC.250.1163
(SEQUENCE-OF-STEPS I-M-AND-GROUP.249.1162
(1 I-M-LOGICAL-DISCONNECT-EVENT.248.1161
(ACTOR I-M-SYSTEM-SM)
(OBJECT1 I-M-GENERIC-RC.422)
(OBJECT2 I-M-CORE-NETWORK.471)
(PRECOND I-M-LOGICALLY-CONNECTED.245.1155
(OBJECT2 I-M-CORE-NETWORK.471))
(ACTION I-M-LOGICAL-DISCONNECT.246.1158
(ACTOR I-M-SYSTEM-SM)
(OBJECT2 I-M-CORE-NETWORK.471))
(RESULT I-M-LOGICALLY-DISCONNECTED.247.1160
(OBJECT2 I-M-CORE-NETWORK.471))))
(PRECOND I-M-PHYSICALLY-CONNECTED.202.1167
(OBJECT1 I-M-GENERIC-RC.422)
(OBJECT2 I-M-CORE-NETWORK.471))
(PRECOND I-M-LOGICALLY-CONNECTED.203.1170
(OBJECT1 I-M-GENERIC-RC.422)
(OBJECT2 I-M-CORE-NETWORK.471))
(FRAMEWORK I-M-SHORT-TERM.610)
(CONFIGURATION I-M-CORE-NETWORK.471)
(CORRECTION-COMPONENT I-M-SYSTEM-SM)
(RESULT I-M-OPERATIONAL.302.1175
(OBJECT I-M-CORE-NETWORK.471))
(FAILED-COMPONENT I-M-GENERIC-RC.422))
(EFFECTED-COMPONENT I-M-CREW.737))
(EFFECT I-M-FAILURE-EFFECT.1273
(EFFECTED-COMPONENT I-M-MISSION-SUPPORT.1009)
(SEQUENCE-OF-STEPS I-M-PROCEDURE.1272
(SEQUENCE-OF-STEPS I-M-NULL-AND-GROUP.86
(1 I-M-NULL-EVENT.85 (ACTION I-M-NULL-ACTION))))
(EFFECT I-M-FAILURE-EFFECT.1282
(EFFECTED-COMPONENT I-M-SYSTEMS.480)
(SEQUENCE-OF-STEPS I-M-PROCEDURE.1272
(SEQUENCE-OF-STEPS I-M-NULL-AND-GROUP.86
(1 I-M-NULL-EVENT.85 (ACTION I-M-NULL-ACTION))))
(EFFECT I-M-FAILURE-EFFECT.1288
(EFFECTED-COMPONENT I-M-INTERFACES.1042)
(SEQUENCE-OF-STEPS I-M-PROCEDURE.1272
(SEQUENCE-OF-STEPS I-M-NULL-AND-GROUP.86
(1 I-M-NULL-EVENT.85 (ACTION I-M-NULL-ACTION))))

5.3 Parsing Input Questions

In FANSYS, the parser operates in two modes called case mode and question mode. The parser behaves identically in each mode with the exception of the degree of indexing that occurs in each. In case mode, every instance of a concept that is created in memory is fully indexed in the generalization indexing hierarchy (in addition to each of the other hierarchies), as described in the section on Memory Creation and Retrieval. Thus, every new instance of a concept will be generalized with other similar instances. This reflects the fact that during case parsing, all

information is useful and should be used to learn about the domain. Since it is the task of the parser to learn from the case texts that it reads, it is desirable that full indexing occur.

In question mode, instances are not generalized. This reflects the different goal that FANSYS has in parsing a question. In case mode, the goal is to learn new concepts and organize them within memory. In question mode, the goal is to formulate a search query and return the results of that query. Questions that are parsed instantiate concepts in memory just as in case mode, but these concepts need not be generalized over, since they are not part of the actual case text. The degree of indexing is the primary difference between the two modes.

It is possible to specify that certain index patterns be active in only either question mode or in case mode. The motivation behind this involves the processing that occurs when a question is parsed. If parsing a manual and rhetorical question is encountered, the system should not activate the search and retrieval patterns that handle that question, since it is not an appropriate context for search and retrieval. In question mode, on the other hand, the system should activate search and retrieval functions if a question is parsed. This is facilitated by making the question patterns with functional processing knowledge operable in only question mode, rather than case mode. In general however, an index pattern will be active in both modes; the phrase "ring concentrator" should certainly activate the same concept in either mode.

Although this dichotomy between the two modes exists, it should be emphasized that processing is identical in all other ways. The flow of control is the same, as are those hierarchies other than the generalization indexing hierarchy, since the same memory structures are parsed into in both cases.

5.3.1 Question Patterns

In parsing questions, there are two primary indexed patterns used. These are shown in figure 5.5, along with sample questions. The parts of the patterns in square brackets are optional. Questions are highly constrained in the format in which they may be posed, primarily for ease of definition, and to provide a well specified means of generating a query. In principle, many different question patterns could be defined to handle the multitudinous ways in which questions are posed by people; this would simply involve adding more lexical patterns to the system to handle such cases. As it stands, the format for queries has been chosen to be similar to the format used by the User Interface point and click mechanism.

What is the (requested-item) [and the (requested-item)] [for the (given-item)] [when the (item) is (specifier)] [and the (item) is (specifier)] *q-mark*
"What is the failure mode for the ring concentrator when the failure mode is loss of output - failure to start ?"
What are the (requested-item) in the (given-item) when the (item) is (specifier) *q-mark*
"What are the detection steps in the next node detection procedure when the failed component is the ring concentrator ?"

Figure 5.5
The Two Question Formats

Once a question MOP is instantiated and activated by recognizing one of the two question patterns, the information in the slots is extracted by an associated function attached to the question MOP. The memory search functions detailed in the section on Memory Creation and Retrieval are then invoked by the question MOP, and control is relinquished to the search module, which performs the search and generates the result.

5.3.2 An Annotated Question Example

The process of parsing a question will now be illustrated by an annotated trace. This trace is heavily edited; it is assumed that the reader is familiar with the parsing mechanism from the earlier trace of the parsing of case RC.1. Only those features specific to the processing of a question will be highlighted.

Parsing

(WHAT IS THE FAILURE CAUSE FOR THE TIME GENERATION UNIT *Q-MARK*).

Reading WHAT

(WHAT * IS THE (REQUESTED M-QUESTION-CONSTRAINT) AND THE
(REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)

WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)

Q-MARK) = M-QUESTION

(WHAT * ARE THE (REQUESTED M-AND-GROUP) IN THE

(GIVEN M-QUESTION-CONSTRAINT) WHEN THE (STATE M-STATE-ASSERTION)

Q-MARK) = M-QUESTION2

As was discussed earlier, there are two query formats. These are stored with the question word "what", and both become active upon parsing that word.

Reading IS

(WHAT IS * THE (REQUESTED M-QUESTION-CONSTRAINT) AND THE

(REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)

WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)

Q-MARK) = M-QUESTION

Reading THE

(WHAT IS THE * (REQUESTED M-QUESTION-CONSTRAINT) AND THE
(REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
Q-MARK) = M-QUESTION

Reading FAILURE

Reading CAUSE

(FAILURE CAUSE *) = M-FAILURE-CAUSE referenced

Specializing: I-M-FAILURE-CAUSE.4416

(WHAT IS THE (REQUESTED M-QUESTION-CONSTRAINT) * AND THE
(REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
Q-MARK) = M-QUESTION

The MOP class M-QUESTION-CONSTRAINT organizes two types of MOPs, systems and procedures. These are the only types of knowledge representation that are allowed to be slots within a question MOP. Since a failure cause is a type of procedure, it satisfies the requirement of the "requested" slot, and the pattern can be advanced. Note that since the parser is in question mode while parsing this question, the MOP I-M-FAILURE-CAUSE.4416 is not indexed in the generalization indexing hierarchy. This prevents a content-less MOP like this one from being used to draw generalizations from text, particularly since that text has no relevance to describing faults in the domain.

Reading FOR

(WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.4416) AND THE
(REQUESTED M-QUESTION-CONSTRAINT) FOR * THE
GIVEN M-QUESTION-CONSTRAINT) WHEN THE (STATE M-STATE-ASSERTION)
AND THE (STATE M-STATE-ASSERTION) *Q-MARK*) = M-QUESTION

Since the "and the..." part of the phrase is optional, the pattern can still be advanced past the point of the word "for".

Reading THE

(WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.4416) AND THE
(REQUESTED M-QUESTION-CONSTRAINT) FOR THE * (GIVEN M-QUESTION-CONSTRAINT)
WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
Q-MARK) = M-QUESTION

Reading TIME

(TIME * GENERATION UNIT) = M-GENERIC-TGU

Reading GENERATION

(TIME GENERATION * UNIT) = M-GENERIC-TGU

Reading UNIT

(TIME GENERATION UNIT *) = M-GENERIC-TGU referenced

Specializing: I-M-GENERIC-TGU.3723

(WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.4416) AND THE
(REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
* WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
Q-MARK) = M-QUESTION

As noted before, systems fulfill the M-QUESTION-CONSTRAINT filler constraint. The time generation unit thus works as a filler for the "given" slot.

Reading *Q-MARK*

(WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.4416) AND THE
(REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN I-M-GENERIC-TGU.3723)
WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
Q-MARK *) = M-QUESTION referenced
Specializing: I-M-QUESTION.4421

The state assertions are also optional elements of the question phrase, so the pattern can be advanced and recognized. Question MOPs have functional code associated with them that interface to the Memory Creation and Retrieval Module. Once the question MOP is recognized, the slots are passed down to Memory Search and Retrieval, where the query is created and processed.

Performing Search:

- [1] ERRONEOUS INPUT
- [2] PIECE-PART FAILURES
- [3] CONTAMINATION
- [4] TEMPERATURE (HIGH OR LOW)
- [5] MECHANICAL SHOCK
- [6] THERMAL SHOCK

Found good match. Elaborate further anyway? (Y/n) n

Once the user has specified that she or he is satisfied with the results of the query, control is returned to the parser, which in this case cleans up any index patterns that were not resolved during the parse.

Removing: I-M-IP.4385.4410
Removing: I-M-IP.479.4418
Removing: I-M-IP.481.4417
Removing: I-M-IP.479.4414
Removing: I-M-IP.481.4413
Removing: I-M-IP.4384.4411

5.4 Generating Answers to Input Questions

Just as the parser maps text into the conceptual structures that underlie it, some mechanism is needed to map memory structures back into natural language. This is the task of the generator. The generator relies on the same stereotypical patterns of language usage that the parser relies on, since most of the same index patterns that are used to parse text are used to generate back into text.

The algorithm for generation is as follows:

1. Choose a MOP to generate.
2. Find the first index pattern with that MOP as its target that satisfies any imposed constraints (such as syntactical constraints).
3. If no such pattern exists, look for a pattern up that MOPs abstraction hierarchy.
4. A. If no pattern is found, simply return the MOP name (e.g. words return the word name).
B. Otherwise, for each element of the pattern, recursively apply this generation algorithm.

As an example, consider the generation of the MOP I-M-TIME-OUT-EVENT.144.507. A portion of the indexing hierarchies characterizing this MOP is shown in figure 5.6. The following is a trace by the generation function which details the execution of the algorithm just described:

```
>(generate I-M-TIME-OUT-EVENT.144.507)
Checking mop I-M-TIME-OUT-EVENT.144.507 for phrases:
  Found: NIL
Checking mop M-TIME-OUT-EVENT.144 for phrases:
  Found: NIL
Checking mop M-TIME-OUT-EVENT for phrases:
  Found: (M-IP.358)
Considering phrase M-IP.358:
  ((1 (M-SYNTAX-CATEGORY RECIPIENT M-SYSTEM)) (2 WILL)
   (3 REACH) (4 A) (5 TIME-OUT) (6 LIMIT) (7 FOR) (8 RECEIPT) (9 OF)
   (10 THE) (11(M-SYNTAX-CATEGORY MESSAGE M-MESSAGE))
   (12 *PERIOD*))
Using phrase: M-IP.358
Checking mop I-M-SYSTEM-SM for phrases:
  Found: NIL
Checking mop M-SM for phrases:
  Found: (M-IP.338)
Considering phrase M-IP.338:
  ((1 SYSTEM) (2 MANAGEMENT))
Using phrase: M-IP.338
Checking mop I-M-TOKEN.491 for phrases:
  Found: NIL
Checking mop M-TOKEN for phrases:
  Found: (M-IP.339)
Considering phrase M-IP.339:
  ((1 NETWORK) (2 TOKEN))
Using phrase: M-IP.339
("SYSTEM" "MANAGEMENT" "WILL" "REACH" "A" "TIME-OUT" "LIMIT" "FOR"
 "RECEIPT" "OF" "THE" "NETWORK" "TOKEN" ".")
```

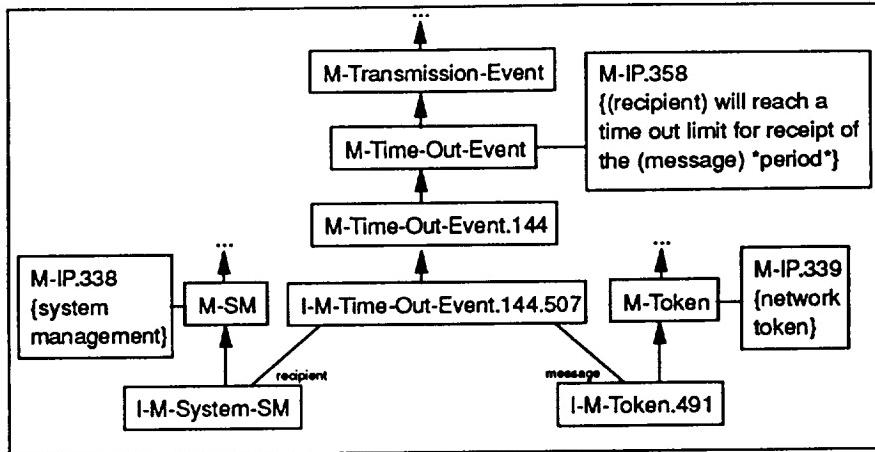


Figure 5.6
Sample Hierarchy for I-M-Time-Out-Event.144.507

One constraint on the generation of patterns is that of syntax. Patterns may be given membership within a syntactic category during pattern definition time using the syntax-category attribute shown previously in figure 5.2. Fillers for slots within patterns may also be syntactically constrained. In order to use a pattern for the generation of a slot filler, it has to be of the same syntactic category as that slot constraint. This allows the generator to generate in a syntactically correct manner, rather than just stringing together syntactically inconsistent phrases. For example, consider another phrase that might have M-TIME-OUT-EVENT as its target: "the timing out of the (system) for the (message)". This phrase might be assigned to a category like VERB-PHRASE. We would not want to use it to generate a complete sentence, since it wouldn't be proper grammatically, whereas the phrase in the annotated trace works fine for this purpose. These constraints can be controlled through the use of syntactic categories. In practice, the index patterns within FANSYS are syntactically unconstrained, since the text has been simple enough so as not to cause problems. Should the lexical domain become more richly enhanced, however, some form of syntactic constraint is needed. The syntactic constraints that can be placed upon a pattern provide for just such measures, should they become necessary.

Although in theory the exact same patterns could be used to generate output that are used to parse text, there are in practice some constraints placed on patterns in terms of whether they should be used only for generation, only for parsing, or both. Most of the patterns in the system are used for both parsing and generation, but a few are modified with :gen and :nogen in the pattern definitions detailed previously in figure 5.2. The :gen specifies that a pattern should be used for generation only, while a :nogen modifier specifies that a pattern should be used only to parse text. An example of a pattern that might be marked with a :nogen modifier is that which characterizes the sentence S1: "indication of a RC failure to start is first detected by the next active node on the network," which is a pointer to the M-NEXT-NODE-DETECTION procedure. (See the detection procedure section of the annotated parse trace for the actual index pattern which is used to characterize S1.) When generating the M-NEXT-NODE-DETECTION procedure, it is preferable to have as much information characterizing that procedure included within the generation; this most certainly includes the steps of the procedure. Generating from the pattern which characterizes S1 would not include that information; it would generate essentially the same sentence as S1. It is therefore marked with the :nogen specifier.

The index pattern that is actually used to generate any sort of procedure (including M-NEXT-NODE-DETECTION) is one similar to IP2: { (1 M-EVENT) (2 M-EVENT) ... } => M-PROCEDURE, where (1), (2), and so on are the actual events that compose the procedure. IP2 generates a list of sentences describing the actual procedure under consideration. This is too abstract of a pattern to be used in parsing procedures, however, since the construction of a procedure is typically more involved than just listing together events (there is some search and recursive construction of the procedure that occurs; see the annotated parse trace for some of the details). IP2 is therefore marked with the :gen modifier, which specifies that it be used only for generation.

Although the :nogen and :gen specifiers exist, they are in practice seldom used, since most of the patterns defined in the system are useful for both parsing and generation. Were a more sophisticated generation algorithm in place, one which considered the space of all available patterns and chose the pattern which best characterized the concept to be generated (rather than just choosing the first pattern it finds), then the :nogen and :gen modifiers could probably be abandoned. Since generation per se has been a secondary concern to the project, a simple generation algorithm with the :gen and :nogen functionality has been chosen to perform generation. Within this limited domain, this approach has proven sufficient for the generation tasks required by FANSYS.

6. Memory Creation and Retrieval

The ISA, Lexical, and Slot-Filler hierarchies described in sections 3 and 4 organize items hierarchically under predefined categories. For example, index patterns are lumped underneath the index pattern MOP, lexical entries are grouped underneath the lexicon MOP, and so forth. In addition, the ISA and Slot-Filler memory hierarchies index cases by only one path that does not contain generalizations across cases. While this organization is useful for the parsing process, it is not an organization which is appropriate for retrieving applicable cases during a question/answering session.

During question/answering FANSYS must be able to quickly retrieve either generalizations of its knowledge, or specific cases from a potentially large database of cases. Our solution to this problem in FANSYS is to overlay an additional indexing structure on top of the ISA memory hierarchy. This additional indexing structure is based upon the memory representation model used in Kolodner's CYRUS system (Kolodner, 1983a, 1983b, 1984). Collectively, these indices form a rich hierarchy of generalizations where many indices point to each case. Since this memory representation allows for indexing and the creation of generalizations, this hierarchy will be referred to as the Generalization-Indexing (GI) memory hierarchy. Although similar to the indices used by Kolodner in CYRUS, the GI indices employed in FANSYS differ in that they allow for attribute values to be represented hierarchically.

In addition to allowing for case retrieval, the GI memory hierarchy allows FANSYS to perform memory tasks similar to those performed by CYRUS, including: (1) generalization across cases, (2) encoding specificity* by grouping similar cases together, (3) fast retrieval by limiting memory traversal to relevant indices, and (4) elaboration strategies which dictate where in memory to search if the initial search query fails. Before discussing the details of the algorithms used in FANSYS for these tasks, the underlying data structures and the method in which comparisons across cases are made must be discussed.

6.1 MOP Attributes

As in the other hierarchies, the fundamental data structure used in the GI memory hierarchy is also the Memory Organization Packet (MOP). To recap, a MOP is a unit which represents a particular concept. By grouping MOPs hierarchically, high-level MOPs represent abstractions of the MOPs below it. There are two types of MOP's: the MOPs located at the bottom of the memory hierarchy are *instances* (denoted I-M-MOPNAME), while the MOPs which organize the instances are *abstractions* (denoted M-MOPNAME).

While the same MOP structure is used in all of the memory hierarchies, the GI memory hierarchy requires the use of several different constructs. The bad-indices of the MOP indicate attributes which do not make good indices. The elaborations of the MOP indicate other MOPs which may aid in memory retrieval. The role of both of these attributes will be discussed in the sections of memory creation and retrieval. The slot of the MOP contains the conceptual definition

* Encoding specificity (Tulving, 1972) refers to the psychological phenomenon of memory case retrieval based upon the features present in the original processing of the case. Case retrieval can only occur if these features have been discriminated.

of the MOP. The norm of the MOP contains generalized information about its sub-MOPs (specializations). Finally, the indices of the MOP contain attribute/value pairs which point to its sub-MOP specializations. In FANSYS, attributes are atomic but values may be defined hierarchically.

Attributes, slots, and the values of indices all use the same list-based tree structure. This representation employs lists which may contain other lists, which may in turn contain other lists. By linking these lists together, detailed tree-like structures can be created. The general format for these lists is $((role\ filler)\ (role\ filler)\ ...)$, where the role is an atomic attribute, and the filler is either an atomic element or recursively defined as another list of the same structure.

To illustrate this format, consider the following example taken from the failure detection procedure for the ring concentrator. In this procedure, a faulty ring concentrator is detected when the next ring concentrator in the network fails to receive a token from the previous ring concentrator. As a precondition for this procedure, the next ring concentrator must be ready to receive a token from the previous ring concentrator. This is represented below as the READY-TO-RECEIVE event, where the RECIPIENT is the next ring concentrator (I-M-NEXT-RC), the SENDER is the current ring concentrator (I-M-RC), and the object being sent is a token (I-M-TOKEN):

```
(PRECOND ((HEADER I-M-READY-TO-RECEIVE)
          (SENDER
            ((HEADER I-M-RC)
             (SOFTWARE
               ((HEADER M-SM)
                (SOFTWARE M-SOFTWARE-COMPONENT)))
               (HARDWARE
                 ((HEADER M-TRANSMITTER)
                  (SOFTWARE M-SOFTWARE-COMPONENT)
                  (HARDWARE M-HARDWARE-COMPONENT))))))
          (RECIPIENT
            ((HEADER I-M-NEXT-RC)
             (SOFTWARE
               ((HEADER M-SM)
                (SOFTWARE M-SOFTWARE-COMPONENT)))
               (HARDWARE
                 ((HEADER M-TRANSMITTER)
                  (SOFTWARE M-SOFTWARE-COMPONENT)
                  (HARDWARE M-HARDWARE-COMPONENT))))))
          (OBJECT I-M-TOKEN)))
```

In this example, the top-level list is of the form:

```
((PRECOND (SENDER (...)) (RECIPIENT (...)) (OBJECT I-M-TOKEN)))
```

The information contained within the ellipses is defined with an identical structure. In this particular example, the SENDER and RECIPIENT are ring concentrators which in turn have two sub-components, a software and a hardware component. The software component is the system management (M-SM), while the hardware component is a transmitter (M-TRANSMITTER). Notice that these components are defined recursively, in that they have their own hardware and software components. If finer detail is required, more components can be added and each component can be further defined in terms of sub-components.

This representation scheme allows FANSYS to accurately represent a variety of devices, actions, events, procedures, or concepts in terms of aggregates or semantic meaning. The slots, norms, and indices of the GI memory hierarchy all use this same format. For instance, if the previous example is a slot, then the conceptual content of the MOP would be the ready-to-receive event for a ring concentrator. If the example is a norm, then a generalization of the MOP would be the ready-to-receive event, and sub-MOPs would represent specializations of ready-to-receive. Finally, if the examples was used in an index, then the ready-to-receive event would serve as the link between another MOP concept.

6.2 Linking MOPs

As described above, the values of indices which link MOPs take the form of detailed lists, where the lists may represent hierarchical concepts. Similarly, the method in which MOPs are linked is also hierarchical. Just like the ISA memory hierarchy, the GI memory hierarchy is also organized in terms of abstractions and instances. Abstract MOPs occupy the internal nodes of the tree, while specific instances occupy the leaves. However, unlike the ISA hierarchy, the GI hierarchy is much more richly indexed. In the GI hierarchy, each instance will be indexed by all possible features of that instance. This results in multiple retrieval paths and "deep learning" of each case.

Indices are internally represented within each MOP by the form: $((Attribute\ Value\ Next-MOP)\ (Attribute\ Value\ Next-MOP)\ ...)$, where Attribute is an atomic feature, Value is a list of the form described in the previous example, and Next-MOP is the name of the MOP which is linked by the index. Using this representation, indices are uni-directional and always point downwards in the hierarchy towards more specialized MOPs.

The rule for linking MOPs together is to use index values which differ from the norms. Cases which are similar to one another will be grouped close together, but these cases will be indexed in the manner which they differ from each other. This indexing strategy allows distinguishing features to be the factor which determines which cases are retrieved. The actual algorithm used to select these indices will be presented in the section on memory creation.

As an example, consider the MOP M-CASE. This MOP organizes all cases stored in memory. Any norms which this MOP may contain apply to all cases in memory. MOPs which are indexed directly below M-CASE are a bit more specialized; MOPs indexed below these MOPs are even more specialized, and so on, until we finally reach the instances at the leaves of the hierarchy. Each MOP contains norms which are the generalizations of all MOPs below it. Furthermore, each MOP is indexed by the way in which it is different from other MOPs. If the

MOPs had the configuration of indices shown below they would encode the sample hierarchy shown graphically in figure 6.1. To keep this example simple only a few indices have been specified. The actual implementation contains many more indices.

M-CASE indices:

```
((FAILED-COMPONENT ((HEADER I-M-RC ...) M-MOP.1)
(FAILED-COMPONENT ((HEADER I-M-GW ...) M-MOP.2)
(FAILURE-CAUSE ((HEADER I-M-THERMAL-SHOCK...) I-M-MOP.3))
```

M-MOP.1 indices:

```
((FAILURE-CAUSE ((HEADER I-M-THERMAL-SHOCK)..) I-M-MOP.4)
(FAILURE-CAUSE ((HEADER I-M-CONTAMINATION)..) I-M-MOP.5)
```

M-MOP.2 indices:

```
((FAILURE-MODE ((HEADER I-M-ERRONEOUS-OUTPUT)..) I-M-MOP.6))
```

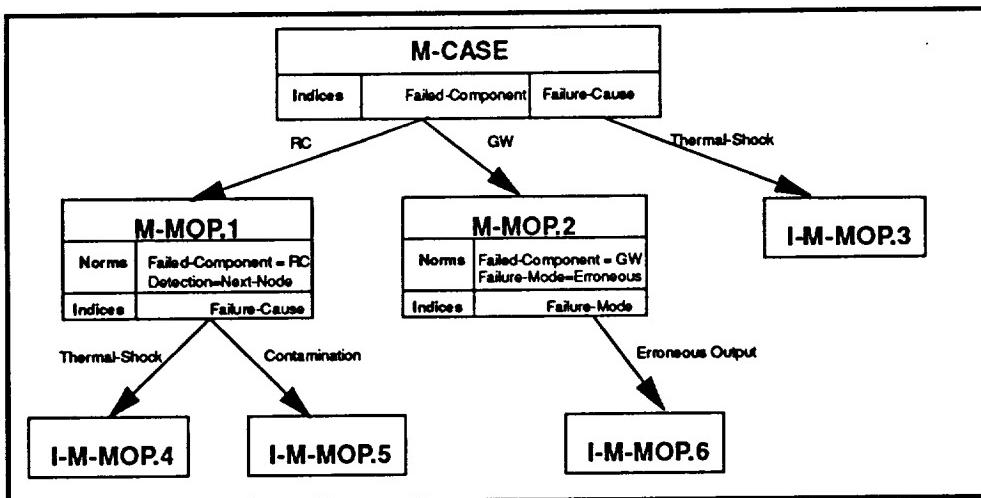


Figure 6.1
Sample GI Memory Hierarchy

In this example, the attribute of FAILED-COMPONENT may take on two different values. MOP.1 is linked to M-CASE if the value of the FAILED-COMPONENT is ring concentrator. MOP.2 is linked to M-CASE if the value of the FAILED-COMPONENT is gateway. I-MOP.3 is linked to M-CASE if the value of the FAILURE-CAUSE is thermal shock. In this example, the ellipses denote that additional details may be specified for the attribute's values, as in the previous example (to specify the hardware components, software components, etc.). Just like the MOP M-CASE, MOPs 1 and 2 have indices of their own pointing to more specialized MOPs. The values of the indices indicate the ways in which MOPs differ from one another. MOPs 1,4, and 5 all involve the failure-component of a ring concentrator. I-MOP's 4 and 5 are further differentiated by the failure-cause. I-MOP 4 is indexed by thermal shock, while I-MOP 5 is indexed by contamination. Finally, generalizations are stored at all sub-MOPs. In this example, I-MOPs 4 and 5 share the failed-device and the failure cause. This generalization is stored as the norms of their common abstraction, MOP 1. Similarly, the norms of MOP 2 contain the generalizations of all sub-MOPs, in this example I-MOP 6.

Unlike the ISA hierarchy, each index in the GI hierarchy has an attribute and a value. As a result, when MOPs are accessed by traversing these indices, we are now accessing MOPs by their meaning. This will greatly speed up the retrieval process by limiting memory traversal to only those indices which correspond to the memory query. Consequently, search will be constrained to only relevant portions of memory. Kolodner describes the attribute/value indices as "locks". In order to access a structure, we must have the "key." In this case, the key corresponds to the value of an index. As more keys become available, more locks can be opened. Analogously, as more features are provided to the system, the greater the probability of accessing a relevant case increases since more indices can be traversed. Kolodner's notes that this behavior mirrors Tulving's encoding specificity hypothesis, as well as several other psychological results (Kolodner, 1983).

It is important to remember that the GI memory hierarchy is not completely separate from the ISA memory hierarchies. Instead, the GI hierarchy is "overlaid" on top of the ISA memory structures. Both the GI hierarchy and the ISA hierarchies share the same root MOPs and the same instance MOPs. The two hierarchies differ in the way the root MOP is connected to the instances. The GI hierarchy uses a rich set of indices, while the ISA hierarchy uses only enumerated indices. This integration is illustrated in figure 3.5 in the system architecture section. An area of future work is to merge these two hierarchies into a single hierarchy which combines features of both memory schemes.

6.3 Comparing Cases

Before discussing the algorithms used for memory creation and retrieval, an important operation which must be understood is how concepts are compared. All slots, indices, and norms are stored in the form of lists specified in the section on MOP attributes. When creating generalizations, adding new MOPs to memory, or determining which indices can be traversed, the comparison operation must be invoked. This operation is made more complicated due to the hierarchical representations of concepts. For example, a standard data processor running a particular program might be represented as ((HEADER I-M-SDP) (SOFTWARE I-M-PROGRAM.1)), while a standard data processor running a different program might be represented as ((HEADER I-M-SDP) (SOFTWARE I-M-PROGRAM.2)). If strict equality were used to compare these two representations, we would end up with the result that the components are not equal. However, both components are standard data processors. A more useful comparison operation would return an answer which indicates how well one item matches with another.

The comparison function implemented in FANSYS returns more meaningful information by computing a percentage of match between two concepts. The algorithm considers which values to compare in a top-down, depth-first manner and then propagates the percent comparison for each value in a bottom-up fashion. An overview of the COMPARE algorithm is described below.

Given a list of attribute/value pairs from a target T,
and a query Q, start at the topmost level:

(0) Set M=0

(1) For each attribute/value pair in Q do:

 If T contains the same attribute then

 If T's value = Q's value set M=M+1

 else if T's value is a list

 Set M=M+ (recurse to step 1 using
 the values of Q and T)

(2) Return M/(Number of attribute/value pairs in Q)

An example of comparing two cases is given below. The target case is a subset of the example described in the previous section on MOP attributes. The query is similar to the target except some of the slots do not exactly match. A sender is specified in the query but is missing in the target, and one of the ring concentrator's software components within the query is different than the target.

Case= ((RECIPIENT
 ((HEADER I-M-NEXT-RC)
 (SOFTWARE
 ((HEADER M-SM)
 (SOFTWARE M-SOFTWARE-COMPONENT)))
 (HARDWARE
 ((HEADER M-TRANSMITTER)
 (SOFTWARE M-SOFTWARE-COMPONENT)
 (HARDWARE M-HARDWARE-COMPONENT)))))
 (OBJECT I-M-TOKEN))

Query= ((RECIPIENT
 ((HEADER I-M-NEXT-RC)
 (SOFTWARE
 ((HEADER M-SM)
 (SOFTWARE M-SOFTWARE-COMPONENT)))
 (HARDWARE
 ((HEADER M-TRANSMITTER)
 (SOFTWARE M-PROGRAM-2)
 (HARDWARE M-HARDWARE-COMPONENT)))))
 (SENDER M-RC)
 (OBJECT I-M-TOKEN))

The case and query are depicted graphically in figures 6.2 and 6.3.

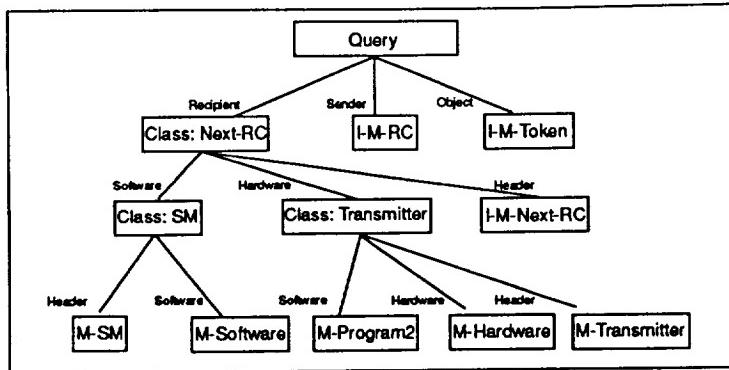


Figure 6.2
Graphical representation of sample query

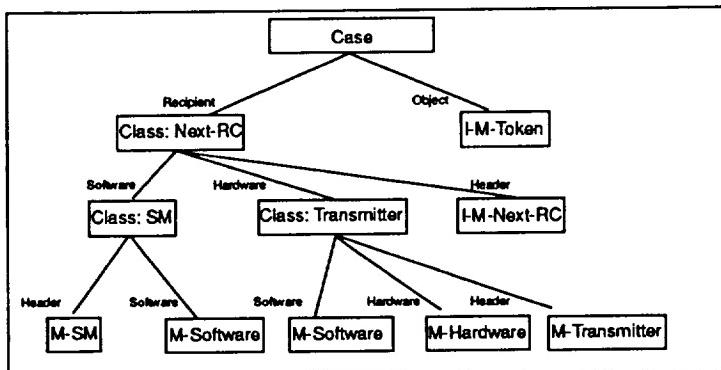


Figure 6.3
Graphical representation of sample case

The top-down stage of the algorithm determines which attributes should be compared. At the top-most level, the query consists of ((RECIPIENT ...) (SENDER ...) (OBJECT ..)). The attributes are RECIPIENT, SENDER, and OBJECT. The algorithm loops through each of these attributes and checks whether or not the case also contains the same attribute. In this example, RECIPIENT is contained within the target case. However, the value does not exactly match the query, and it is a list, so M will be set to M+X, where X is some value determined recursively via Compare((HEADER ..) (SOFTWARE ..) (HARDWARE..)). The next attribute is SENDER. However, the target contains no sender attribute, and so nothing is done. The last attribute is OBJECT. The target does have an OBJECT attribute, and the value matches the query exactly, so M is set to M+1. When the algorithm is finished, M=X+1, and the ratio (X+1)/3 will be returned. Values for the entire computation of the query are shown below, starting from the bottom-up. 1's indicate matches, and ?'s indicate values which haven't been computed yet.

```
((RECIPIENT ?
  ((HEADER 1)
  (SOFTWARE ?
    ((HEADER 1)
    (SOFTWARE 1))))
  (HARDWARE ?)
  ((HEADER 1)
  (SOFTWARE 0)
  (HARDWARE 1)))))

(SENDER 0)
(OBJECT 1))
```

Percentages propagate up from the leaves:

```
((RECIPIENT
  ((HEADER 1)
  (SOFTWARE 2/2)
  ((HEADER 1)
  (SOFTWARE 1)))
  (HARDWARE 2/3)
  ((HEADER 1)
  (SOFTWARE 0)
  (HARDWARE 1)))))

(SENDER 0)
(OBJECT 1))
```

These values propagate up again:

```
((RECIPIENT
  ((HEADER 1)
  (SOFTWARE 1)
  (HARDWARE 2/3)))
  (SENDER 0)
  (OBJECT 1))
```

The final match percentage is given as: $(((1+1+2/3)/3) + 0 + 1) / 3 = 0.63$

The backed-up values are depicted on the graphical representation of the query shown below in figure 6.4. The horizontal slashed line for "Sender" indicates a comparison which is never made since there is no Sender slot in the case.

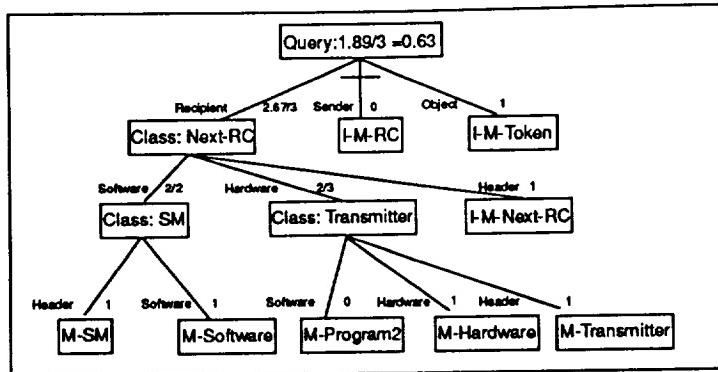


Figure 6.4

Comparing case from figure 6.3 against query from figure 6.2. Numerical values on the right indicate backed-up values. The horizontal line indicates a comparison which is never made since the attributes do not match.

Notice that this process compares the case against the query. If the query is compared against the case, then a different match percentage would be returned.

The percentage returned by the comparison function indicates that the case matches the query with 63% accuracy. While this number indicates how well features of the cases match, it doesn't include any measure of which features are more relevant than others. This problem requires additional semantic knowledge not directly addressed in the current implementation of FANSYS.

An algorithm similar to the one used to compare cases is also used to determine the similarities and the differences between cases. Determining similarities and differences are important in the memory creation process to create new generalizations and new indices.

Computing the similarities between cases uses almost the exact same algorithm used to compare cases, except equivalent values are saved as the algorithm runs. Computing the similarities between the case and query of figures 6.2 and 6.3 yields:

```
Sim = ((RECIPIENT
        ((HEADER I-M-NEXT-RC)
         (SOFTWARE
          ((HEADER M-SM)
           (SOFTWARE M-SOFTWARE-COMPONENT)))
         (HARDWARE
          ((HEADER M-TRANSMITTER)
           (HARDWARE M-HARDWARE-COMPONENT))))))
        (OBJECT I-M-TOKEN))
```

Any attributes or values which are not present in both cases is simply removed from the end result. Finding similarities will be used in determining the norms of MOPs so that generalizations may be created.

Finally, computing the differences between cases also follows a similar algorithm to the comparison algorithm. However, when values of the query and case do not match, the case values are stored and eventually returned. Just like the comparison algorithm, computing the differences of case 1 vs. case 2 is different from computing the differences of case 2 vs. case 1. When two values aren't the same, what should be returned, the mismatch from case1 or the mismatch from case 2? The approach taken in FANSYS is to use the differences with respect the second case. Typically, the function to compute the differences will be called twice, once to get the differences for case 1 and again the get the differences for case 2.

In our example with the query and target case, computing the differences between the case and query with respect to the query returns mismatches from the query:

```
Dif = ((RECIPIENT  
        (HARDWARE  
          ((SOFTWARE M-PROGRAM2))))  
        (SENDER M-RC))
```

Similarly, computing the differences between the query and the case with respect to the case returns mismatches from the case:

```
Dif = ((RECIPIENT  
        (HARDWARE  
          ((SOFTWARE M-PROGRAM2)))))
```

The attribute/value pair of (SENDER M-RC) is not included since this isn't part of the case, but part of the query.

6.4 Memory Creation Process

With the underlying data structures defined and the algorithms for comparison, computing differences, and computing similarities developed, these components can now be combined into the memory creation process. As described earlier, the GI memory hierarchy must be organized so that MOPs are indexed by their differences, similar cases are grouped together, and the norms of MOPs hold the generalizations of all cases below it. These tasks are discussed by Kolodner in her work on long-term memory (Kolodner, 1983).

To accomplish these tasks within FANSYS, the GI memory system must first be given a case to index. This case will be some type of instance with a set of slots and attribute/value pairs which describe the conceptual content of the case. These instances are generated as the parser processes some text. This means that each instance created by the parser will also be passed to the GI memory system for indexing. In addition to passing the GI memory system a case, the parser will also pass a MOP where indexing should begin. For instances of entire cases this would be M-CASE (the MOP which organizes all cases), for components this would be M-ORU, and so on.

Once a case and a MOP location to begin indexing has been determined, the process of memory creation can begin. First, the case is separated into its attribute/value pairs. At this point, we would ideally like to weed out any irrelevant attributes and retain only the distinctive attributes. This is the difficult feature-selection problem. In the current implementation of FANSYS, non-predictive features have been preselected and stored in the bad-index slot of the MOP. Any attributes which are marked as bad-indices are then discarded. For complete cases, the failure-mode, failed-component, failure-detection procedure, and the failure-correction procedure are all predictive attributes, but the failure-cause and the failure-effects are nonpredictive since virtually all cases share the same values for these attributes. By stripping the nonpredictive attributes, memory will be less cluttered and more meaningful. However, note that it is not crucial to remove these attributes; if they are not removed FANSYS will still function normally, but will have larger memory requirements. While the current implementation of FANSYS requires these attributes to be specified by the designers, future work includes learning nonpredictive attributes automatically through pre-processing or a real-time statistical analysis or clustering of the data.

After relevant indices have been selected, the similarities between the case and the norms of the current MOP are computing using the similarity function. The norms of the MOP are then set to these similarities. This guarantees that the norm will only hold generalized information which applies to all cases which have been processed by the current MOP.

The next step is to traverse any indices whose attribute and value match an attribute and value of the new case. To determine if the index matches an attribute, the previously defined comparison function must be invoked. If the index and the attribute's value match greater than some threshold value, then the index is traversed. In FANSYS, this threshold was arbitrarily set at 0.5. Further experiments are necessary to determine if other threshold values are more appropriate. In general, a low threshold value will make indices very easy to traverse. For example, a threshold of 0 would result in all indices being traversed if the values had anything at all in common, no matter how small. This will result in much generalization, but also more indices will be created leading to potential memory problems. A high threshold value will require a very close match and make indices harder to traverse. While fewer indices would be created, less generalization would be made between cases.

By only traversing matching indices, traversal is restricted to relevant indices. If no indices match, then a new index is created which points to the new case. If matching indices do exist and the indexed MOP is a generalization, then the entire procedure is repeated recursively using the new MOP as the entry point. However, if the indexed MOP is a specific instance, then the differences between this instance and the new instance are computed, and these two cases are then indexed by their differences.

By traversing to the end of the hierarchy along matching indices, we will end up grouping similar cases together. Ultimately, cases are indexed by their differences, so the unique features of a case will be the crucial indices which lead to a case. Notice that this entire process applies to all matching indices at every step of the computation; this results in the creation of many redundant indices - each case will be indexed by all features of the case itself.

The algorithm controlling the memory creation process is as follows:

0. Given a new instance case to add and a starting MOP:
1. Select:
 - A) Features to use as indices
 - B) Remove features noted as bad indices
2. Create new Instance-MOP with slots of the new case
3. Start at the given entry MOP:
 - A) Set norms to similarities between norms, case
 - B) For all attribute/value pairs in the new case:
 - 1) If matching index from the current MOP doesn't exist, create one pointing to the new case.
 - 2) Else if index exists to M-MOP (abstraction), recurse with the new MOP as the root (step 3), with the case attributes minus the attribute/value pairs which have already been processed
 - 3) Else if index exists to Instance-MOP then:
 - Create new M-MOP
 - Calculate similarities between instance, new mop.
 - Add similarities as norms to new M-MOP.
 - Calculate differences between instance, new mop.
 - Index MOPS based on differences from new M-MOP.

An example of the memory creation process is shown in figures 6.5 and 6.6. For each case which is added to memory, the non-predictive attributes are removed and a new instance of the case is created. For entire cases, the entry MOP will be M-Case; i.e., all cases will be indexed under the Case MOP. Next, indices which match the new instance will be traversed. When collisions with other instances occur, new indices will be created based upon the differences between the two instances. When the first case is added to memory, there are no indices from M-Case to traverse. Consequently, the new case is simply indexed by all the attributes of the case. This is shown in figure 6.5 for I-M-RC.1, the ring concentrator case where the Failure Mode is Startup-No-Output. In this figure, the indices of the Failure Mode, Failed Component, and Failure Correction procedure are shown. Indices are also created for the Failure Cause, Detection Procedure, and the Effect, but these are not shown.

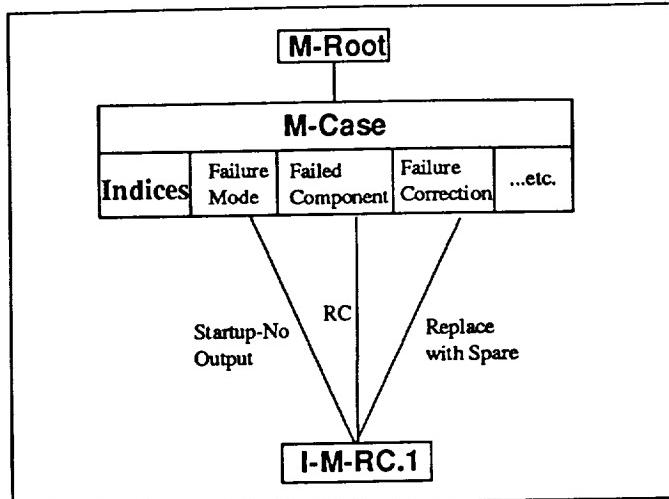


Figure 6.5

GI hierarchy after adding a single ring concentrator case to memory. Only the failure mode, failed component, and failure correction links are shown here.

Figure 6.6 shows how memory organizes itself when a second case is added to memory. In this figure, I-M-RC.1 has already been processed when I-M-GW.1 is given to the system. This new case involves the gateway rather than the ring concentrator. Both cases have the same Failure Mode of Startup-No-Output, and the same long-term Failure Correction Procedure of Replace-with-Spare, but the cases differ in the component which has failed. When I-M-GW.1 is processed, matching indices will be traversed until no more indices exist or a collision occurs. For example, since both cases share the index of Failure Mode, this index will be traversed. A collision will then occur at I-M-RC.1. A new sub-mop, M-Generalization.1 will be created, and the generalizations between I-M-RC.1 and I-M-GW.1 are computed and used as the norms for M-Generalization.1. Next, the differences between I-M-RC.1 and I-M-GW.1 are computed and both instances are indexed from the new generalization based upon these differences. In this example, the cases differ in the Failed Component, so indices of the different failed components are created to access the two instances.

A similar process occurs by traversing the Failure Correction Procedure index, resulting in M-Generalization.2. Since only two cases have been processed, M-Generalization.2 will be identical to M-Generalization.1. However, these MOPs will be different as more cases are processed and different generalizations are computed. Finally, new indices are also created from M-Case for a Failed Component equal to the Gateway, since this index value does not currently exist. This entire process would also occur for the indices which are not shown in the figure (e.g., Failure Detection Procedure, Failure Cause).

In addition to organizing cases according to their differences, the memory creation algorithm also generalizes across all cases and their components. In the example shown, a generalization has been made between a case involving the ring concentrator and a case involving the gateway; both cases share the same Failure Mode and Failure Correction Procedure. By generalizing across all cases, FANSYS is capable of processing generalized questions and also returning generalized answers. Moreover, FANSYS may also discover similarities not obvious to humans reading the same cases.

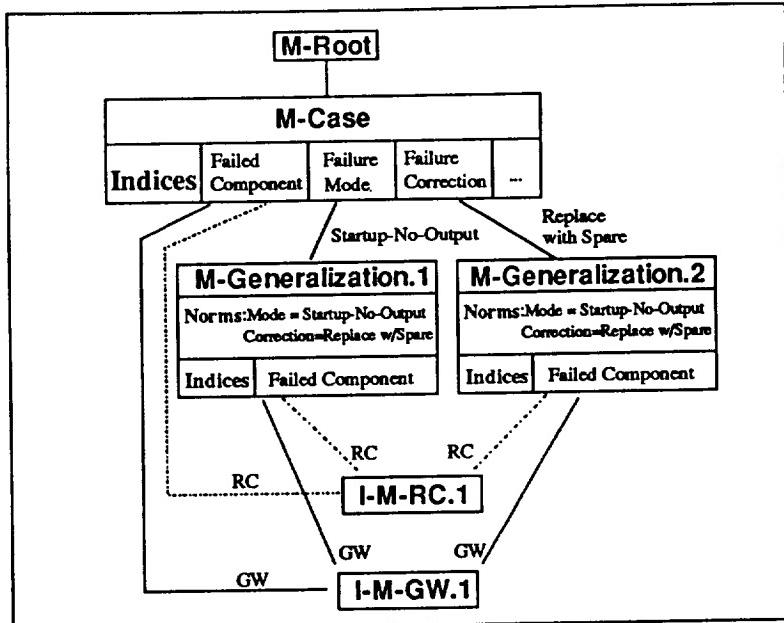


Figure 6.6

GI hierarchy after adding a ring concentrator and gateway case to memory. The generalization MOPs are created to norms holding information common to both cases. The cases themselves are indexed by their differences. In this example, only the failed component is shown. Other indices may be created for the detection procedure, other correction procedures, etc.

An advantage of this indexing scheme is that memory is *self-organizing*. Cases are automatically added to memory such that similar cases are grouped together yet still separated by their differences. Furthermore, cases are indexed by all attributes which compose the case. This allows cases to be retrieved based upon any related aspect of the case rather than mere surface features or predefined keywords. Psychologists refer to this as "deep learning." Furthermore, these indices may be traversed directly, resulting in fast retrieval. Finally, the hierarchical memory structure allows generalizations to be built across all similar cases.

A potential pitfall of this memory scheme is the problem of exponential explosion. As cases with more features are added to the hierarchy, the number of indices created grows exponentially. This was not a problem in the existing implementation of FANSYS since the pre-determined selection of good and bad indices resulted in a streamlined memory representation. However, this would become a serious problem if a large number of cases were added to the system. A solution to this problem is described by Kolodner in her work with long-term memory (Kolodner, 1983). Her solution controls exponential growth by making generalizations. If a MOP indexes a majority of the same cases that are also indexed by its parent, then this MOP is simply removed, and its norms added to the norms of the parent. This eliminates an entire subtree of the hierarchy, effectively controlling the number of MOPs and indices.

Implementation techniques could also be used to cut down on memory costs. The current representation uses expanded lists within slots, indices, and norms. Many of these lists are identical, particularly indices. The same index value may be used in many different indices within many different MOPs. If only one copy of each expanded index is stored and pointers are used

to reference this definition in lieu of the expanded definition, then there would be a large savings in the amount of occupied memory.

6.5 Memory Retrieval Process

The basic retrieval process is simplified once a memory hierarchy has been created using the algorithm previously described in section 6.5. First, a query must be presented to the system. A query consists of *given* data (variables whose values are provided), and *requested* data (variables whose values are unknown.) The value of the requested data is what the user would like to know. The given data simply consists of a set of attribute/value pairs, and the requested data an attribute without a value. When FANSYS is provided with a query consisting of attribute and value pairs, GI indices are compared to these attributes and values, and matching indices are traversed. Any instances which are reached by the traversal process are returned. Based upon the structures formed by the memory creation process, all indices which point to instances are features of the instances. Since only indices which match the search query are traversed during retrieval, all instances that are retrieved must be relevant to the search query and these instances are accessible without enumeration of indices. Kolodner discusses the benefits of this approach in her work on reconstructive memory (Kolodner, 1983).

The current implementation of FANSYS traverses matching indices in a depth-first manner. The depth-first search is not performed among all indices, but only indices which match the input. This drastically reduces the number of nodes which must be searched, making retrieval very fast. In effect, the attribute/value indices prevent the enumeration of indices and instead allows indices to be content-addressable. This is a desirable property, particularly when a large number of cases need to be represented in memory. On average, the retrieval time will remain constant with respect to the number of cases which are added to memory. Consequently, retrieval time will not suffer when scaling up to large knowledge bases.

Retrieval is also facilitated by the rich indexing scheme. Each case is indexed by all features of the case. This allows case retrieval based upon any distinguishing features. For example, many hierarchical systems may index a case about the ring concentrator only under the category of ring concentrators. This works fine as long as these cases are retrieved in the context of ring concentrators. However, if one wants to retrieve these cases based upon other features (such as the failure-detection method, the failure-mode, etc.) then retrieval is much more difficult because the system has to figure out that the ring concentrator index applies to the new query. In FANSYS, each case is indexed by all features of the case, and the problem of finding appropriate indices is avoided.

Before any indices may be traversed, a suitable entry-point in the parsing hierarchy must be selected where search can begin. A suitable entry-point would be a MOP which subsumes all of the data given in the question. The motivation for choosing such a MOP is that the answer is probably indexed somewhere under this MOP if attributes of the query and the requested information are also indexed under this MOP. While this may not always be true, this assumption

does provide a good place to begin search. As an example, if the goal of the system is to retrieve an entire case, search will typically begin at M-CASE, the MOP which organizes all cases. On the other hand, if the goal of the system is to retrieve an orbital replacement unit, then M-ORU, the MOP which organizes all ORU's, is a logical place to begin search.

The strategy used in FANSYS to select the entry point is to use the MOP which is an abstraction of all data provided in the query. This process begins by starting at the MOPs representing the requested data. If one of these MOPs is an abstraction of the given data and the requested data, then that MOP is used as the entry point. If no MOPs organize all the known data, then the process is repeated with the immediate abstractions of the MOPs compared in the previous step. The first MOP which organizes all data provided in the question is used as the entry-point. If no entry-point is found, the entire process is repeated starting from the MOPs representing the given data. This procedure finds the most specific MOP which also subsumes the given and requested data in a bottom-up fashion. Note that this entire process is computed using the ISA and Slot-Filler hierarchy, not the GI hierarchy.

Once the entry point has been determined, the GI indices which match the input query are traversed and all instances found are returned. The basic retrieval algorithm is outlined below:

- 0) Supplied a list of requested items and given items:
- 1) Find an appropriate entry point to begin searching.
This is the most specific MOP which is an abstraction of the given and requested data. Find this MOP searching from the bottom-up beginning at the MOPs representing the requested or the given data.
- 2) Try direct retrieval
Traverse all indices where the attribute/value match an attribute/value of the query (the given data).
If a generalization is reached, repeat the process starting at the new MOP (go back to step 2).
If an instance is reached, save it.
- 3) Compare all retrieved cases to the input query, and return those which match 100% or within some user-specified percentage.

As an example of this process, consider the query, "What is the failure correction procedure for the gateway when the failure mode is loss of output - failure to start?" An annotated memory search and retrieval trace for this query is shown below:

Question: What is the failure correction procedure for the gateway when the failure mode is loss of output - failure to start?

Given Data: (MODE M-STARTUP-NO-OUTPUT)
(FAILED-COMPONENT M-GW)
Requested Data: (M-FAILURE-CORRECTION)

Memory search and retrieval is first given the attributes and values representing the question. In this example, the user is asking for the failure correction procedure (M-FAILURE-CORRECTION) where the mode is loss of output - failure to start (M-STARTUP-NO-OUTPUT) and the failed component is the gateway (M-GW). The parsing step is not shown here.

Searching bottom-up for Entry-MOP based on requested:

Entry-MOP=M-CASE

The system has found an Entry-MOP by searching bottom-up from the requested data (M-FAILURE-CORRECTION). The most specific MOP which subsumes M-FAILURE-CORRECTION, M-GW, and M-STARTUP-NO-OUTPUT) is M-CASE. The algorithm can now begin by searching for GI indices from M-CASE which match the query.

Performing Direct Search. Indices:

((MODE M-STARTUP-NO-OUTPUT)
(FAILED-COMPONENT M-GW))

Current MOP: M-CASE

Input matches indices to: (COMP.2499 COMP.2982)

By traversing the index MODE=M-STARTUP-NO-OUTPUT from M-Case, MOP COMP.2499 can be accessed. This MOP generalizes across all cases where the Mode is Startup-No-Output. The index FAILED-COMPONENT=M-GW leads to MOP COMP.2982. This MOP generalizes across all cases where the Failed Component is the Gateway.

Current MOP: COMP.2499

Input matches indices to: (I-M-GW.1)

Current MOP: I-M-GW.1

Current MOP: COMP.2982

Input matches indices to: (I-M-GW.1)

Found: (M-CASE COMP.2499 COMP.2982 I-M-GW.1)

The traversal process is repeated from each sub-MOP of M-CASE. Both paths lead to the same instance, I-M-GW.1, which is the first gateway case. One path goes from M-CASE to COMP.2499 to I-M-GW.1 by traversing the index MODE-M-STARTUP-NO-OUTPUT from M-CASE to COMP.2499, and then the index FAILED-COMPONENT=M-GW from COMP.2499 to I-M-GW.1. The other path traverses a failed component index first, and then the mode index second to access the case. Note that although the multiple paths to an instance may seem redundant, this indexing scheme allows generalizations to be built across components, failure modes, failure correction procedures, etc. The traversal path for this example is depicted graphically in figure 6.7. Only the paths which are actually traversed are shown; all other indices are not considered since the attributes and values do not match those provided in the query.

After traversal has finished, all instances which completely match the input are returned and the portions of the case which match the requested data are generated in English.

Percent Match of Values to Query: ((M-CASE 0) (COMP.2499 0) (COMP.2982) (I-M-GW.1 1))

Picking out those that match 100%

Results of Direct Search: (I-M-GW.1)

- [1] SHORT TERM CORRECTION PROCEDURE : THE SYSTEM MANAGEMENT SELECTS A BACKUP GATEWAY TO REPLACE THE GATEWAY FROM A LOOK-UP TABLE MAINTAINED WITHIN THE DMS . THE SYSTEM MANAGEMENT POWERS DOWN THE GATEWAY . THE SYSTEM MANAGEMENT POWERS UP THE BACKUP GATEWAY . CORRECTION PROCEDURE : THE SYSTEM MANAGEMENT LOGICALLY DISCONNECTS THE GATEWAY FROM THE DMS NETWORK . THE SYSTEM MANAGEMENT LOGICALLY CONNECTS THE BACKUP GATEWAY TO THE DMS NETWORK .
- [2] LONG TERM CORRECTION PROCEDURE : THE CREW PHYSICALLY DISCONNECTS THE GATEWAY FROM THE DMS NETWORK . THE CREW PHYSICALLY CONNECTS THE BACKUP GATEWAY TO THE DMS NETWORK . THE CREW LOGICALLY DISCONNECTS THE GATEWAY FROM THE DMS NETWORK . THE CREW LOGICALLY CONNECTS THE BACKUP GATEWAY TO THE DMS NETWORK .

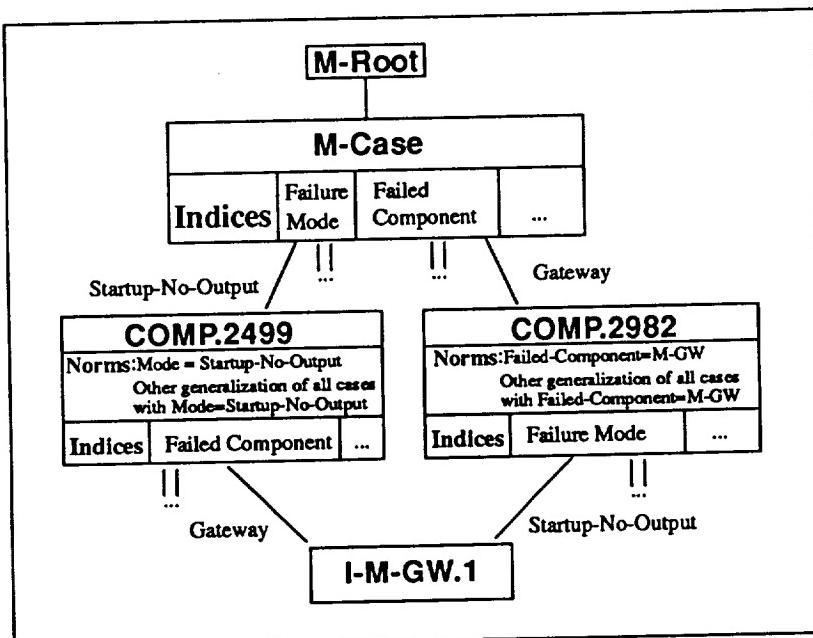


Figure 6.7

Traversal paths for example query given Failure-Mode=Startup-No-Output and Failed-Component=M-GW starting from Entry MOP=M-CASE

A second trace is shown below. In this example, the user is asking FANSYS to specify details of the lookup-backup correction procedure:

Question: What are the correction steps for the lookup backup procedure when the failed component is the standard data processor?

Given Data: (FAILURE-CORRECTION M-LOOKUP-BACKUP)
(FAILED-COMPONENT M-SDP)

Requested Data: (M-CORRECTION-AND-GROUP)

Searching bottom-up for Entry-MOP based on requested:

M-OR-GROUP

M-AND-GROUP

No MOP found which subsumes given data

Starting from the requested data, a MOP was not found which subsumes all data provided in the question. By performing another search from the given data, a suitable MOP is found. This MOP turns out to be M-LOOKUP-BACKUP, one of the MOPs actually specified by the user. As a result, there will be no indices of FAILURE-CORRECTION=M-LOOKUP-BACKUP starting from M-LOOKUP-BACKUP, but there will be indices for the failed component.

Searching bottom-up for Entry-MOP based on given:

Entry MOP=M-LOOKUP-BACKUP

Performing Direct Search. Indices:

(FAILURE-CORRECTION M-LOOKUP-BACKUP)
(FAILED-COMPONENT M-SDP)

Current MOP: M-LOOKUP-BACKUP

Input matches indices to: (I-M-LOOKUP-BACKUP.3061)

Current MOP: I-M-LOOKUP-BACKUP.3061

Found: (M-LOOKUP-BACKUP I-M-LOOKUP-BACKUP.3061)

Percent Match of Values to Query: ((M-LOOKUP-BACKUP 0) (I-M-LOOKUP-BACKUP.3061 1))

Picking out those that match 100%

:

Results of Direct Search: (I-M-LOOKUP-BACKUP.3061)

- [1] THE SYSTEM MANAGEMENT SELECTS A BACKUP STANDARD DATA PROCESSOR TO REPLACE THE STANDARD DATA PROCESSOR FROM A LOOK-UP TABLE MAINTAINED WITHIN THE DMS . THE SYSTEM MANAGEMENT POWERS DOWN THE STANDARD DATA PROCESSOR. THE SYSTEM MANAGEMENT POWERS UP THE BACKUP STANDARD DATA PROCESSOR. CORRECTION PROCEDURE : THE SYSTEM MANAGEMENT LOGICALLY DISCONNECTS THE STANDARD DATA PROCESSOR FROM THE DMS NETWORK . THE SYSTEM MANAGEMENT LOGICALLY CONNECTS THE BACKUP STANDARD DATA PROCESSOR TO THE DMS NETWORK.

In this example, we are not retrieving an entire failure case instance, but an individual instance of the lookup-backup failure correction procedure. As illustrated in figure 6.8, this instance is indexed directly under M-LOOKUP-BACKUP by traversing only a single index. By finding an appropriate entry-point MOP to begin search, the number of MOPs which need to be examined can be drastically reduced.

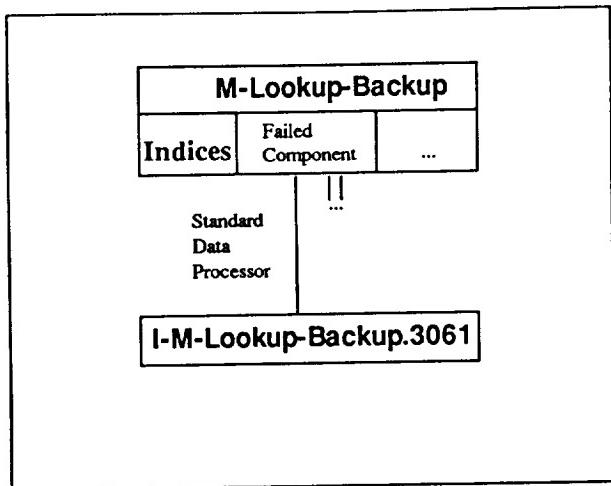


Figure 6.8

Traversal path for example query given Failed-Component=Standard Data Processor with the Entry MOP=M-Lookup-Backup

The basic retrieval algorithm works well when the QA module is given enough information to traverse indices all the way down to the instances. However, the system should also be capable of handling more difficult questions where the input may not give enough information to access a specific case. One solution is to *elaborate* on the input query. In general, elaboration is the process of generating related items to use for search based upon a particular input query. For example, when asked to name all 50 states of the United States, many people will elaborate on the question by thinking of states they have visited, states where friends or relatives live, postal abbreviations, or perhaps by visualizing a map and mentally traversing the states. None of these retrieval techniques were directly specified by the input query, but were conjured up by searching for a context where an appropriate case may be found.

While Kolodner specifies a number of elaboration techniques in her work on memory retrieval (Kolodner, 1983), the current implementation of FANSYS uses only a set of simple elaboration strategies. The first strategy removes some of the constraints of search by allowing all indices to be traversed which have the same attributes as the requested information. This amounts to adding to the search query all possible requested items. By allowing more indices to be traversed, there is a greater chance of retrieving an instance. Furthermore, the new indices which are traversed are related to the input question, so we are still restricting search to relevant portions of memory. As an example, if we are searching for a ring concentrator, then all ring concentrator indices will be traversed even if the value doesn't match an attribute/value given in the query. This is often just what is needed to retrieve cases, since the requested item is typically not known and hence not present in the query. This elaboration technique is called *elaborating on the requested information*.

If elaborating on the requested information still fails to retrieve cases, then the search can be broadened by allowing even more indices to be traversed. This is accomplished in FANSYS by retrieving the attributes which are contained in the *elaborations* slot of each MOP traversed. The information in the elaborations slot specifies additional attributes and other MOPs where search may be conducted. Any indices which have these attributes are allowed to be traversed, even if the value of the index does not match attribute/values of the query. Currently,

the elaborations are pre-defined in FANSYS to include the failure-mode, failure-detection procedures, and failure correction procedures. A topic for future work would include methods of learning appropriate MOPs and attributes for elaboration.

Finally, if the above two elaboration strategies still fail to retrieve any applicable cases, then a depth-first-search of the entire memory can be employed, retrieving all cases. Each case can then be matched with the query, and the most relevant returned. With a large memory, this will be a time-consuming process and should be done only as a last resort. In our experiments, a complete search of memory was never necessary.

The retrieval process, accounting for elaboration, is summarized below:

- 1) Try basic retrieval
- 2) If no matches found, allow elaboration on the requested data. Run the retrieval algorithm again, but allow traversal of any indices matching the requested data.
- 3) If no matches are found, allow elaboration on all attributes contained in the elaborations slot of each MOP traversed. Run the retrieval algorithm again, but allow traversal of any indices matching these new attributes. Also run the retrieval algorithm starting at all MOPs specified in the elaborations slot.
- 4) If no matches are found, perform a depth-first-search and retrieve all instances.
- 5) Compare all retrieved cases to the input query, and return those which match 100% or within some user-supplied percentage. This could also be modified to return the highest N matches.

An example of retrieval with elaboration is shown in the trace below. In this example, the user is asking for the failed component when the heartbeat detection procedure is used. The only information given in the query is the detection procedure. This is not enough information to access cases. FANSYS will have to elaborate and traverse indices not specified in the question.

Question: What is the failed component when the failure detection procedure is the heartbeat detection procedure?

Given Data: (FAILURE-DETECTION M-HEARTBEAT-DETECTION)
Requested Data: (M-ORU)

Searching bottom-up for Entry-MOP based on requested:
Entry-MOP=M-CASE

Performing Direct Search. Indices: (DETECTION M-HEARTBEAT-DETECTION)
Current MOP: M-CASE
Input matches indices to: (COMP.4325)
Current MOP: COMP.4325
Found: NIL
Values: NIL

COMP.4325 is the MOP which generalizes across all heartbeat detection procedures; i.e., it is accessed through the index DETECTION=HEARTBEAT-DETECTION.

Picking out those that match 100%

Results of Direct Search: NIL

None found.

No cases are retrieved since the single index of DETECTION=HEARTBEAT-DETECTION is not sufficient to access a case. Consequently, the system gets "stuck" at COMP.4325. The traversal is depicted in figure 6.9 in the encircled area. FANSYS will next lessen the search constraints by allowing traversal of any indices which have an attribute of FAILED-COMPONENT. This is accomplished by performing the search again with a new wildcard added to the search query.

Elaborating using wildcard for requested information.

* matches with anything. Indices:

(DETECTION M-HEARTBEAT-DETECTION)

(FAILED-COMPONENT *)

Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE

Input matches indices to: (COMP.4199 COMP.3569 COMP.2992 COMP.1321 COMP.4325)

Current MOP: COMP.3569

Current MOP: COMP.1321

Current MOP: COMP.4199

Input matches indices to: (I-M-SDP.2)

Current MOP: I-M-SDP.2

Current MOP: COMP.2982

Input matches indices to: (I-M-GW.2)

Current MOP: I-M-GW.2

COMP.3569 and COMP.1321 respectively organize cases for the time generation unit and the ring concentrator. Since the query has been expanded to traverse any index with a FAILED-COMPONENT both of these MOPs will be examined. However, none of these MOPs have indices which match the query, so traversal halts. However, COMP.4199 organizes all cases for the standard data processor. This MOP does have indices which match the input data of DETECTION=M-HEARTBEAT-DETECTION which lead to the instance I-M-SDP.2. Similarly, the MOP which organizes all gateway cases also leads to an instance, I-M-GW.2.

Current MOP: COMP.4325

Input matches indices to: (I-M-GW.2 I-M-SDP.2)

Current MOP: I-M-GW.2

Current MOP: I-M-SDP.2

Before elaboration, FANSYS was only able to traverse to COMP.4325, the generalization for HEARTBEAT-DETECTION. However, the system is now capable of reaching instances due to the expanded search query. All failed-component indices are traversed, leading to the GW and

SDP instances previously found. These instances are returned and the requested information generated from the retrieved cases. The paths traversed before and after elaboration are shown in figure 6.9.

Found: (COMP.4325 I-M-GW.2 M-CASE COMP.4199 COMP.2982 I-M-SDP.2)

Percent Match of Values to Query:

((COMP.4325 0) (I-M-GW.2 1) (M-CASE 0) (COMP.4199 0)
 (COMP.2982 0) (I-M-SDP.2 1))

Picking out those that match 100%

Results of elaboration: (I-M-GW.2 I-M-SDP.2)

- [1] GATEWAY
- [2] STANDARD DATA PROCESSOR

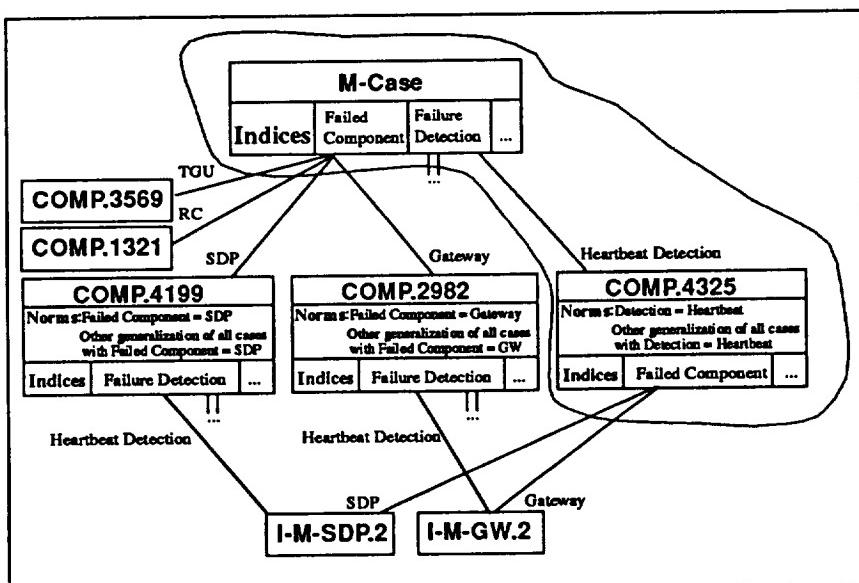


Figure 6.9

Traversed path for example query with elaboration given Failure-Detection=Heartbeat Detection with the Entry MOP=M-Case. The encircled area represents MOPs traversed before elaboration. After elaboration all paths shown are traversed.

A final annotated trace depicting a case where FANSYS uses a last resort strategy of depth-first search along all unspecified indices is shown below. In this example, the system is given a single failure mode, failure during operation with erroneous output, and is asked to return all of the failure causes associated with this mode. The given information and requested information alone are not enough to reach any specific instances since many cases share the same causes when the mode is erroneous output. Consequently, FANSYS must expand upon unspecified indices to access cases.

Question: What is the failure cause when the failure mode is failure during operation - erroneous output?

Given Data: (FAILURE-MODE M-OPERATIONAL-ERRONEOUS-OUTPUT)
 Requested Data: (M-FAILURE-CAUSE)

Searching bottom-up for Entry-MOP based on requested:
Entry-MOP=M-CASE

Performing Direct Search. Indices: ((MODE M-OPERATIONAL-ERRONEOUS-OUTPUT))

Current MOP: M-CASE

Input matches indices to: (COMP.1902)

Found: NIL

Values: NIL

Picking out those that match 100%

Results of Direct Search: NIL

None found.

Direct search traverses the single index MODE=M-OPERATIONAL-ERRONEOUS-OUTPUT from M-CASE to COMP.1902 and then stops. Since no instances are reached, elaboration using the requested information is employed, allowing FANSYS to traverse any CAUSE indices.

Elaborating using requested information.

* matches with anything. Indices: (MODE M-OPERATIONAL-ERRONEOUS-OUTPUT)
(CAUSE *)

Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE

Input matches indices to: (COMP.1326 COMP.4376 COMP.1325 COMP.1324
COMP.1902 COMP.1323 COMP.1322)

Current MOP: COMP.1326

Current MOP: COMP.4376

...

Current MOP: COMP.1323

Input matches indices to: (I-M-GW.3)

Current MOP: I-M-GW.3

Current MOP: COMP.1322

Input matches indices to: (I-M-TGU.3)

Current MOP: I-M-TGU.3

Found: (COMP.1326 COMP.1325 COMP.1324 COMP.1323 COMP.1322 I-M-GW.3 M-CASE
COMP.4376 I-M-TGU.3)

Values: ((COMP.1326 0) (COMP.1325 0) (COMP.1324 0) (COMP.1323 0) (COMP.1322 0)
(I-M-GW.3 1) (M-CASE 0) (COMP.4376 0) (I-M-TGU.3 1))

Picking out those that match 100%

Results of elaboration: (I-M-GW.3 I-M-TGU.3)

[1] PIECE-PART FAILURES

[2] ERRONEOUS INPUT

Found good match. Elaborate further anyway? (Y/n) Y

After allowing traversal of all CAUSE indices, additional MOPs are traversed and two instances are returned. These two instances are specified uniquely by the failure causes of Piece-Part Failures and Erroneous Input. However, additional cases may exist which are not retrieved by the first elaboration strategy. To retrieve these cases, FANSYS can elaborate upon all unspecified slots, illustrated in the final section of the trace below:

Elaborating on unspecified entry slots...

Indices: ((FAILED-COMPONENT *) (MODE M-OPERATIONAL-ERRONEOUS-OUTPUT)
(CAUSE *) (DETECTION *) (CORRECTION *) (EFFECT *))

Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE

Input matches indices to: (COMP.3569 COMP.2499 ... COMP.1902)

Current MOP: COMP.3569

 Input matches indices to: (I-M-SDP.1 I-M-SDP.2 I-M-SDP.3)

Current MOP: I-M-SDP.1

Current MOP: I-M-SDP.2

Current MOP: I-M-SDP.3

Current MOP: COMP.2499

 Input matches indices to: (I-M-GW.1 I-M-GW.2 I-M-GW.3)

....

Found: (COMP.3569 I-M-RC.3 I-M-TGU.1 COMP.4376 ...)

Values: ((COMP.3569 0) (I-M-RC.3 1) (I-M-TGU.1 2591/2940) (COMP.4376 0) ...)

Picking out those that match 100%

Results of secondary elaboration: (I-M-SDP.3 I-M-GW.3 I-M-RC.3 I-M-TGU.3 I-M-TGU.3)

[1] PIECE-PART FAILURES

[2] ERRONEOUS INPUT

[3] CONTAMINATION

[4] TEMPERATURE (HIGH OR LOW)

[5] MECHANICAL SHOCK

[6] THERMAL SHOCK

In the final section of the trace, redundant sections have been removed for brevity. Only the first matching index from M-CASE to COMP.3569, is expanded. This index leads to all cases involving standard data processors. The other indices are traversed similarly. By allowing traversal of unspecified indices, FANSYS is able to retrieve all matching cases. In this example, additional failure causes are found which were not returned from elaboration using only the requested information. However, a large number of indices are traversed which do not lead to relevant cases. For example, the trace indicates that I-M-SDP.1 and I-M-SDP.2 are examined. These cases do not match the search criteria since the failure mode is not erroneous-output, and are subsequently not returned. Traversal to these irrelevant cases is time-consuming and inefficient. Although only a few general questions require this last resort depth-first search strategy for a complete answer, future work for FANSYS includes the elimination of this last resort strategy and the inclusion of alternate context search and more intelligent elaboration strategies.

7. User Interface

The prototype version of FANSYS was primarily designed to experiment with new paradigms in knowledge representation and the modeling of conceptual processes. As a result, user interface issues have received secondary priority. An implementation version of FANSYS would certainly require an additional emphasis placed upon the user interface. User surveys, a design rationale, task analysis, and user testing are all important phases of interface design which needs to be performed. (Blattner, 1992; Gentner & Grudin, 1990; MacLean et. al, 1991). Results from any of these phases can significantly affect the development of the final system.

While the user interface employed in FANSYS is currently in the preliminary stages of development, it does address a number of design issues. First, FANSYS utilizes the X window system through the GARNET UIMS developed at Carnegie Mellon University (Myers, 1991). This allows for integration with other unix-based applications and provides a foundation for future development. Second, an emphasis has been made to simplify the question/answering sessions since this is anticipated to be the most frequently used aspect of the system. This emphasis is manifested through the use of point-and-click buttons for input and a natural language generator for output. Third, the capability to parse questions in natural language has been provided. Finally, a graphical browsing tool has been developed to aid knowledge engineers in examining and modifying memory.

The user interface of FANSYS is composed of the desktop shown in figure 7.1. There are two main windows. The top window is a trace window which maintains a record of the processes used by FANSYS during text comprehension and question answering. This window is used by FANSYS for output of technical information only. The bottom window is an input/output window which displays input case descriptions, input questions, and answers to these questions. This is the primary window; users unfamiliar with the underlying model of FANSYS may elect to hide the trace window and work only with the input/output window. The overall control of the program is handled through a button panel located in the lower right hand corner of the screen. This panel allows the user to enter a search query, inspect memory with a graphical browser, get help about the system, or exit the system.

The first action that must be performed with the user interface is to parse textual descriptions of cases to build up the knowledge base, as described in section 5. As these case descriptions are parsed, the trace window displays the processes involved in understanding the input text. A sample of this process is shown in figure 7.2, which shows the behavior of the system as it begins to parse the first ring concentrator case.

Once the knowledge base has been constructed, question answering may begin. In a typical session the cases will be parsed offline and the knowledge base stored. A user may then start FANSYS directly in question/answering mode. Queries are given to the system in either of two ways: constructing a query through a series of point-and-click buttons, or entering a query via natural language. The point-and-click buttons are simple enough for a novice to use and also gives the user an idea of what types of questions may be asked. Alternately, the natural language interface allows for more types of queries to be created and creates a context in which dialogue is possible.

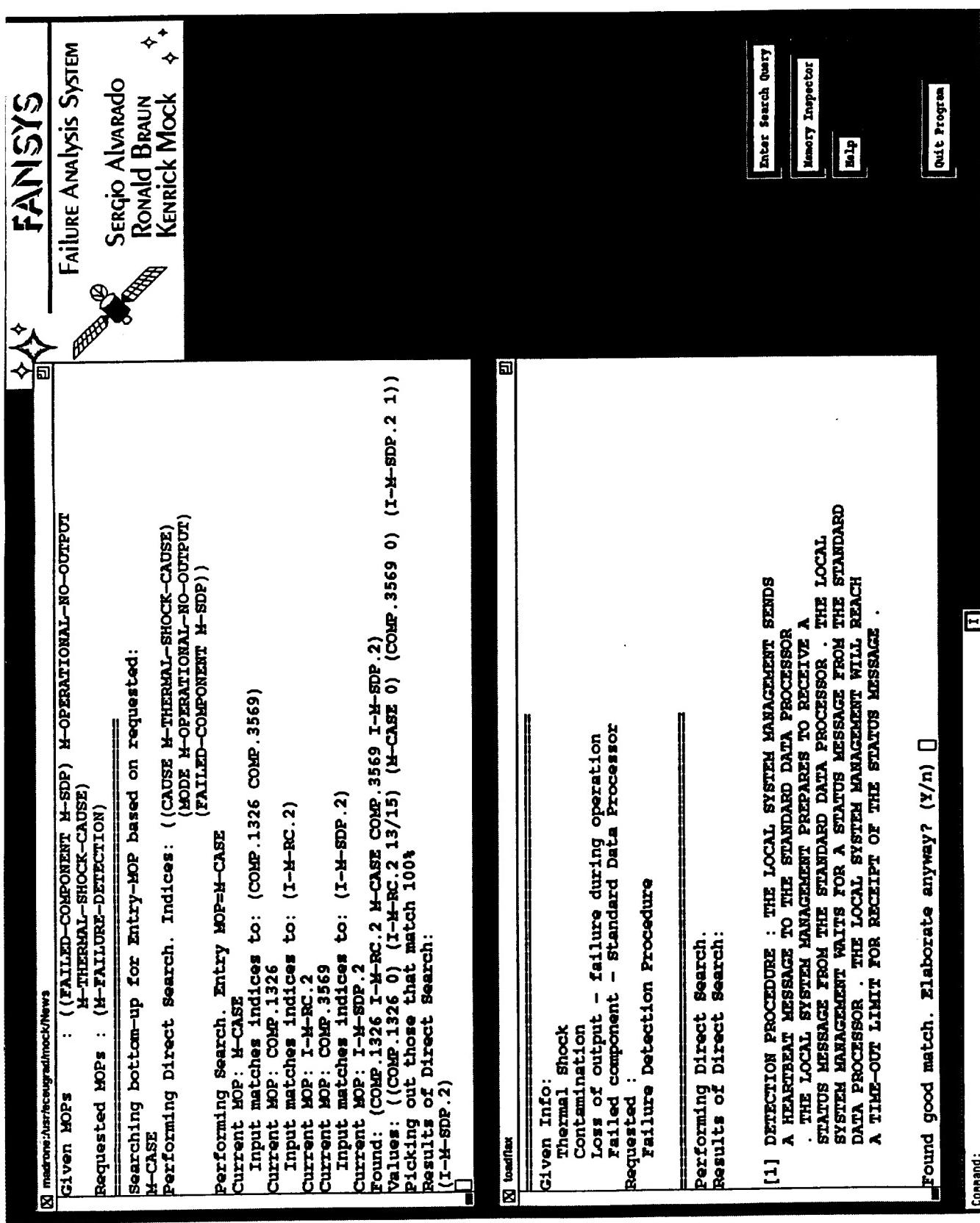


Figure 7.1
FANSYS User Interface. The top window contains a trace of the processes used by the system. The bottom window is the main input/output window. Functions are invoked by the button panel at right.

FANSYS

Failure Analysis System

Sergio Alvarado
Ronald Braun
Kenrick Mock

Parsing (ITEM NAME *COLON* RING CONCENTRATOR FAILURES MODE *ACOLON* LOSS OF OUTPUT - FAILURE TO STATE FAULTS CURES *ACOLON*

PIECE-PART FAILURES *ACOLON* HIGH OR LOW ALERTS-PARENT

TEMPERATURE ALERTS-PARENT HIGH OR LOW ALERTS-PARENT

ACOLON* MECHANICAL SHOCK *ACOLON* THERMAL SHOCK FAILURE DETECTION *ACOLON* VERIFICATION *ACOLON* INDICATION OF A BC FAILURE IS FIRST DETECTED BY THE *ACOLON* NEXT *ACOLON

ACTIVE MODE ON THE NETWORK *ACOLON* LOCAL SYSTEM MANAGEMENT WILL REACH A TIME-OUT LIMIT FOR RECEIPT OF THE NETWORK TOKEN *ACOLON* CONSECUTIVE ACTIONS *ACOLON*

RIGHT-PAREN* A *RIGHT-PAREN* SHORT TERM *ACOLON* NETWORK RECONFIGURATION IS EXECUTED AUTOMATICALLY *ACOLON* THE DUE NETWORK RELOADS IN OPERATION *ACOLON* A RECONFIGURED SOURCE WITH THE FAILED BC BYPASSED *ACOLON* *LEFT-PAREN* 3 *RIGHT-PAREN* LONG TERM *ACOLON* THE CREWMAN CHECK FOR A SOURCE APPLIED POWER *ACOLON* AND THE CREWMAN CHECK FOR *ACOLON* CONNECTOR FIRTHNESS *ACOLON* *RIGHT-PAREN* IF THE BC CANNOT BE PLACED IN OPERATION *ACOLON* THE BC IS REMOVED AND REPLACED WITH AN OLD LOGISTICS SPARE *ACOLON

FAILURE EFFECT ON *ACOLON* *LEFT-PAREN* A *RIGHT-PAREN*

CREW ALERTS STATE *ACOLON* NONE *ACOLON* THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A BC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES

ACOLON* *LEFT-PAREN* 3 *RIGHT-PAREN* MISSION SUPPORT *ACOLON* NONE *ACOLON* *LEFT-PAREN* C *RIGHT-PAREN* SYSTEM *ACOLON* NONE *ACOLON* *LEFT-PAREN* D *RIGHT-PAREN

INTERFACES *ACOLON* NONE *ACOLON* *ACOLON*

Reading ITEM

Recognizing: M-IP.390
Creating: I-M-IP.390.418
Specializing: I-M-IP.390.418
Recognizing: I-M-TERMINATOR.323
(ITEM * NAME *COLON*) = M-ITEM-CONTENT

Reading NAME

(ITEM NAME * *ACOLON*) = M-ITEM-CONTENT

Reading *COLON*

(ITEM NAME *COLON*) = M-ITEM-CONTENT referenced
Recognizing: M-ITEM-CONTENT
Creating: I-M-ITEM-CONTENT.419
Specializing: I-M-ITEM-CONTENT.419
Recognizing: M-CONTENT
Creating: I-M-CONTENT.420
Specializing: I-M-CONTENT.420
Removing: I-M-CONTENT.420
Recognizing: M-CONTENT
Creating: I-M-CONTENT.421
Specializing: I-M-CONTENT.421
Removing: I-M-CONTENT.421

Reading RING

Recognizing: M-IP.349

Command:

Enter Search Query
Memory Inspector
Help
Exit Program

Figure 7.2

Trace information as FANSYS parses the first ring concentrator case. The trace window has been enlarged so that the entire case is visible. Due to spatial considerations, only the first few steps of parsing are shown.

The point-and-click interface is invoked by clicking on the button labeled "Enter Search Query." This will bring up a sequence of button menus which will allow the user to create a query. This query is constructed in terms of "given information" and "requested information". For example, to learn what the failure cause may be for the ring concentrator when the failure mode is startup-no-output, the user will click on the button "Ring Concentrator" to indicate the component which has failed, and "Startup-No-Output" to indicate the failure mode. Together, this data comprises the given information. Finally, the user will click on the button "Failure Cause" to indicate the requested information. Examples of the button menus are shown in figures 7.3-7.9.

Figure 7.3 is the first pop-up window which appears after the user has clicked on the "Enter Search Query" button. This window consists of a number of radio-style buttons which allows the user to select the unknown variables (i.e., the requested data) in the query. In this example, the user is asking the system to determine the failure mode and the failure cause. These menus give the user the opportunity to select or deselect buttons, so any action is easily "undone." When the user is satisfied with the selected choices, the "Continue" button brings up the next menu which prompts the user to enter the given information.

The next menu shown in figure 7.4 prompts the user to enter the component which has failed. If the component is now known, the user may click on the button labeled "Unknown." This menu is always presented to the user since it is anticipated that most queries will contain a known component which has failed. Currently, FANSYS allows the user to select only a single failed component. Upon selecting a component, the menu shown in figure 7.5 appears.

Figure 7.5 depicts a controlling menu which allows the user to give any further information about the search query which may be known. If the failure mode is known, then the user would click upon the "Failure Mode" button. Similarly, if the failure causes are known, then the user would click upon the "Failure Cause" button. Each button will bring up a separate menu allowing the user to specify further data. These menus are shown in figures 7.6-7.9. Note that when only one choice is appropriate, the interface only allows one choice to be made. For example, the user may not specify a query which has two different failure modes since no case has two modes (only an "and" of the input is currently supported; attributes may not yet be "or'd" together). When multiple choices are appropriate, as in specifying the failure causes of figure 7.7, a radio-style button panel is presented to allow many choices to be made. After the user has specified further information, the menu shown in figure 7.5 will re-appear, allowing the user to specify even more information or start the search.

Select the Requested Items

- ORU
- Failure Mode
- Failure Cause
- Failure Effect
- Failure Detection Procedure
- Failure Correction Procedure
- *Detection Steps
- *Correction Steps

Click to Continue

Figure 7.3

Input of search query via point-and-click buttons. This menu allows the user to toggle the items which FANSYS should retrieve.

Select the Failed Component

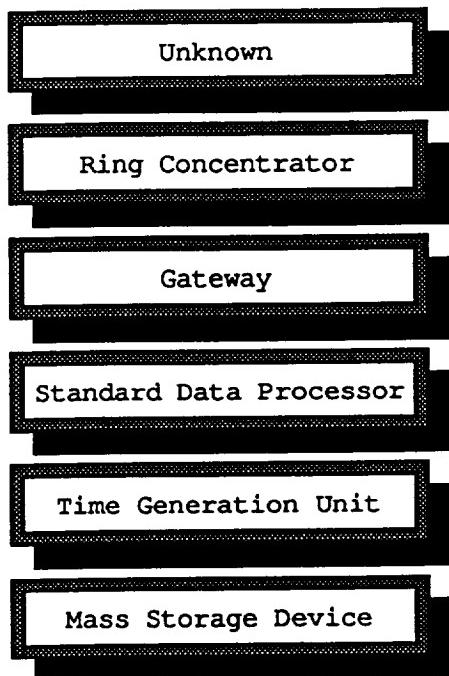


Figure 7.4

Input of search query via point-and-click buttons. This menu prompts the user to select the component which has failed, or "unknown" if the failed device is not known.

Click on the information given in the question



Figure 7.5

Input of search query via point-and-click buttons. This menu allows the user to give any additional information which may be known. After the information has been entered, the "Start Search" button instructs FANSYS to begin searching memory for matching cases.

Select the Failure Mode

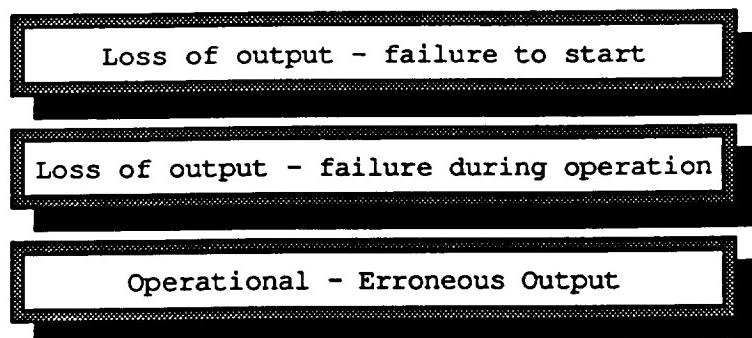


Figure 7.6

Input of search query via point-and-click buttons. This menu prompts the user to give the failure mode. Upon selecting a mode, control returns to the menu shown in figure 7.5.

Select the Failure Cause

- Thermal Shock
- Piece-part Failures
- Contamination
- Temperature (High or Low)
- Mechanical Shock
- Erroneous Input

Figure 7.7

Input of search query via point-and-click buttons. This menu prompts the user to give the failure cause. Upon selecting causes, control returns to the menu shown in figure 7.5.

Select Failure Detection

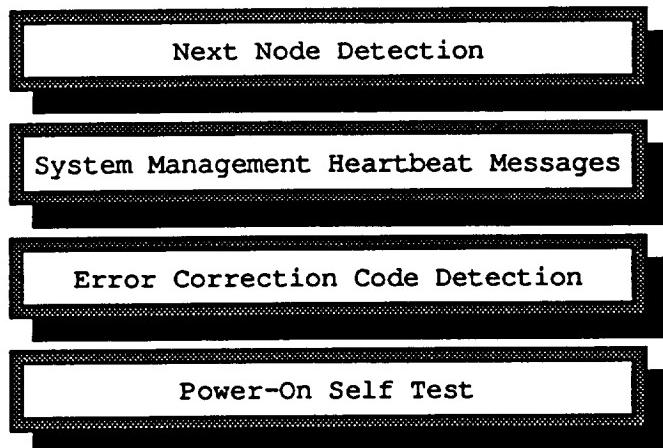


Figure 7.8

Input of search query via point-and-click buttons. This menu prompts the user to give the failure detection procedure. Upon selecting a detection procedure, control returns to the menu shown in figure 7.5.

Select Failure Correction

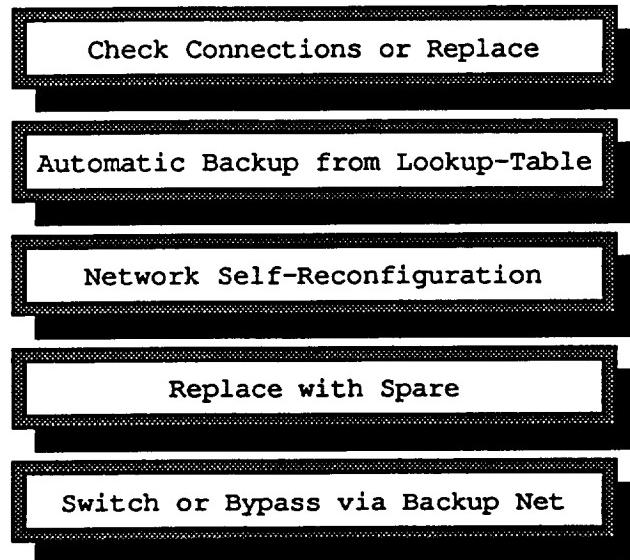


Figure 7.9

Input of search query via point-and-click buttons. This menu prompts the user to give the failure correction procedure. Upon selecting a correction procedure, control returns to the menu shown in figure 7.5.

Once all the information has been given to the system, the button "Start Search" instructs FANSYS to search memory for relevant cases which conform to the query using the retrieval strategies discussed in section 6. A trace of the processing steps is shown in the top trace window, while the query and answer is shown in the bottom question/answer window. Figure 7.1 shows the final result of a sample query where the user has asked FANSYS to determine the detection procedure when the failure cause is thermal shock and contamination, the failure mode is loss of output - failure during operation, and the failed component is the standard data processor. The detection procedure corresponding to the case matching the query is output in the bottom question/answer window via a natural language generator. After cases have been retrieved, the user will be asked if the system should stop or continue searching for more cases which may match the input query.

In addition to using buttons to create a search query, a button also exists to give the user help. This button brings up a scrollable window which describes the overall system, how to use the interface to answer questions, and how queries should be constructed. Future versions of FANSYS will incorporate context-sensitive help. A sample of the user help window is shown in figure 7.10. This particular snapshot shows the format for posing questions to the system via natural language through the question shell.

The question shell is an alternative method of creating queries by allowing users to enter questions in English. As described in section 5, questions are constrained by the format in which they may be posed. Refer to figure 7.10 for a description of the question format enforced by the system. In this figure, items within parentheses denote variables, where several different values may fill that variable. For example, the "given-item" may be "ring concentrator", "gateway", or any other component which may fail. The brackets denote optional patterns which may repeat. For example, a user may ask "What is the failure mode and the failure cause..." to ask the system to return many different answers.

Once a question has been entered, it is processed in order to convert punctuation into the appropriate lexical symbols used by FANSYS (for example, a "." becomes "*PERIOD*"). The sentence is then passed to the parser, which processes it just like any other piece of text (e.g., a case description). Once the question MOP corresponding to the query has been built as described in section 5, the query is constructed and is then passed to the memory search and retrieval module. The retrieved answers are generated in English, just as with the X-Windows interface. An important point worth noting is that the query that is generated is identical to that generated by the X-Windows interface. The only difference is the manner in which the information within the query is specified -- via a point and click mechanism, or via natural language. For instance, figure 7.11 shows the input/output behavior of FANSYS when a question is being processed after it has been typed into the question shell. (This question is the same question used in section 5.3 to illustrate the input/output behavior of FANSYS when parsing a question.) The top window is a trace of the parsing process for the question, while the bottom window shows the results of the query after the memory search module has retrieved the answer.

Help



Questions may also be entered in textual format.
All questions should be of the following form:

what is the (requested-item) [and the
(requested item)] [for the (given-item)]
[when the (item) is (specifier) [and the
(item) is (specifier)]] ?

Note that the question mark must be separated from
the last word by a blank space.



Figure 7.10

FANSYS Help window. This scrollable window provides information about FANSYS and how to use the system to answer questions. This snapshot shows the question format recognized by the question shell.

FAINSYS
 Failure Analysis System


 SERGIO ALVARADO
 RONALD BRAUN
 KENRICK MOCK

```

  matrone/justice/gadmock/News
  Reading TIME
    Recognizing: M-IP.448
    Creating: I-M-IP.448.5642
    Specializing: I-M-IP.448.5642
    Recognizing: I-M-TERMINATOR.437
    (TIME * GENERATION UNIT) = M-GENERIC-TGU

  Reading GENERATION
    (TIME GENERATION * UNIT) = M-GENERIC-TGU

  Reading UNIT
    (TIME GENERATION UNIT *) = M-GENERIC-TGU referenced
    Recognizing: M-GENERIC-TGU
    Removing: I-M-GENERIC-TGU.5643
    Specializing: I-M-GENERIC-TGU.3723
    (WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.5596) AND THE
     (REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
     WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
     "Q-MARK") = M-QUESTION
    Removing: I-M-IP.448.5642

  Reading "Q-MARK"
    (WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.5596) AND THE
     (REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN I-M-GENERIC-TGU.3723) WHEN
     (REQUESTED M-QUESTION-CONSTRAINT))
  
```

testfix

[I] **WHAT IS THE FAILURE CAUSE FOR THE TIME GENERATION UNIT "Q-MARK".**

Parsing (WHAT IS THE FAILURE CAUSE FOR THE TIME GENERATION UNIT "Q-MARK").
 Performing Direct Search.
 Results of Direct Search:
 None Found.
 Elaborating using requested information.
 Results of elaboration:

[1] ERRONEOUS INPUT
 [2] PIECE-PART FAILURES
 [3] CONFIRMATION
 [4] TEMPERATURE (HIGH OR LOW)
 [5] MECHANICAL SHOCK
 [6] THERMAL SHOCK

Found good match. Elaborate further anyway? (Y/n) []

Command: [I]

Enter Search Query
 Memory Inspector
 Help
 Quit Program

Figure 7.11
 Sample of the Question Shell. Parsing is shown in the top trace window, while the parsed question and the answer is shown in the bottom input/output window.

The final component of the user interface is a graphical browser called the Memory Inspector. The Inspector allows a knowledge engineer to inspect and modify memory. The user may click on boxes which represent MOPs. The boxes show the contents of each MOP and the indices connected to other MOPs. By simply clicking on MOPs and indices, memory can be traversed and the hierarchy visually inspected. A sample of the Inspector is shown in figure 7.12. This figure shows the MOP representing M-CASE in the upper-left corner. The user may then choose to expand on any slots or fillers of M-CASE. Clicking upon "SPECS" brings up all specializations of M-CASE; i.e., all of the cases in memory. These are shown in the scrollable window below the MOP of M-CASE. The user may further expand upon one of these cases by simply clicking on it. In the figure, I-M-RC.1, the first ring concentrator case, has been expanded and its slots shown in the upper-right corner. In addition to providing a convenient way to examine and visualize memory, any of these values may also be modified by clicking on them and typing in new values.

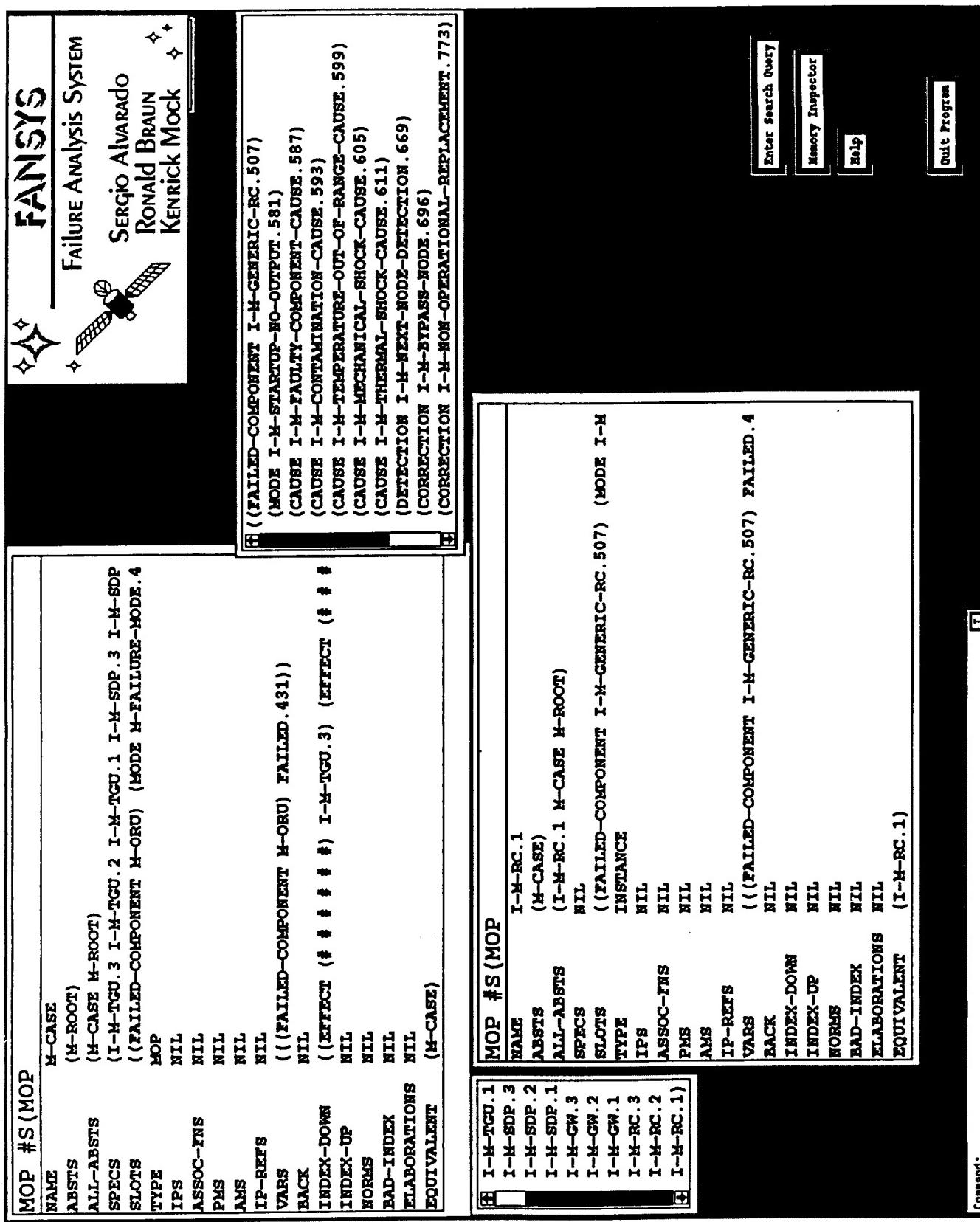


Figure 7.12

Memory Inspector. Clicking upon any slot or filler expands that item in a new window. These values may then be edited. The sample screen depicts the expansion of the first ring concentrator case.

8. Current Status

All of the material discussed in the previous sections has been implemented in a prototype version of FANSYS developed at the U.C. Davis AI lab on Sun workstations. Although only a prototype, a significant effort has been spent in implementation. The source code, written in the CMU Common Lisp programming language, occupies approximately 234K of disk space. The data files used to represent the domain occupies approximately 75K. CMU Common Lisp is a public domain Common Lisp implementation under development at Carnegie Mellon University and is available by anonymous FTP from lisp-rt1.slisp.cs.cmu.edu. Although FANSYS was developed using CMU Common Lisp, the core routines for parsing and memory search/retrieval will run under any version of Common Lisp (e.g., Kyoto Common Lisp, Ibuki Common Lisp, or Franz Allegro Common Lisp). However, some of the user interface routines for interprocess control are dependent upon CMU Common Lisp. The graphical user interface routines require the GARNET User Interface Management System. GARNET is another public domain product developed at CMU and is available by anonymous FTP from a_gp.cs.cmu.edu. GARNET requires the CLX X Library package to be installed on the Lisp platform used.

The current implementation of the parser is capable of reading the first ring concentrator case where the failure mode is loss of output - failure to start. The case text shown in figure 2.1 is an abstraction of the original failure case description. The abstracted case has been further converted to the format shown in the figure; this conversion allows FANSYS to process punctuation more easily.

While the current implementation of the parser has only been developed to parse the ring concentrator case, a total of twelve cases have been hand-coded to test the knowledge representation and memory search/retrieval schemes. These cases cover the descriptions for the ring concentrator, gateway, standard data processor, and the time generation unit. To input these cases, the underlying knowledge to understand the cases is first developed. Next, instantiated knowledge structures representing the cases are presented to the system. These cases are shown in Appendix A. FANSYS then instantiates these knowledge structures and creates GI indices to access the cases for question/answering.

After the twelve cases have been loaded, FANSYS occupies approximately 10 Mb of memory. The core of CMU Common Lisp and the GARNET and CLX interface code occupies another 15 Mb, bringing the total memory requirements to 25 Mb. While the memory requirements are large, little effort has been done to optimize the time and space efficient of the code. More efficient memory representation implementation, particularly in the GI indices where redundancy is high, would result in dramatic memory compaction.

9. Future Work

The goal of FANSYS has been to design a system capable of integrating knowledge constructs with processing strategies to comprehend textual input and answer questions in the domain of failure analysis within the Data Management System of Space Station Freedom. The prototype system has demonstrated the feasibility and power of the approach described in this report. However, as work has progressed, many areas have been discovered where FANSYS can be improved. These areas include the user interface, integration with other tools for fault management, integrating memory structures used for parsing and retrieval, the scope of the implementation, investigating the use of parallel machines, machine acquisition of domain-specific knowledge and search strategies, incorporating model-based and functional-based reasoning strategies, generating belief inferences, and subjective comprehension and verification of input text.

9.1 User Interface

The point-and-click buttons of the user interface currently help make the system intuitive and easy to use. However, the interface has been built upon the system architecture. A deployable system will need to be based upon the tasks required by the users. This may require redesigning portions of the interface and the system itself. Furthermore, additional functionality needs to be designed into the interface so that more complex editing and more flexible question queries can be posed to FANSYS.

9.2 Integration with Other Tools

Additional flexibility can be obtained by integrating FANSYS with other failure analysis tools. For example, FANSYS could provide explanations in natural language to users of NASA's FEAT (Failure Environment and Analysis Tool) (Iverson and Patterson-Hine, 1990; Patterson-Hine and Iverson, 1990; Stevenson, Miller, and Austin, 1991). FEAT provides excellent reasoning capabilities for determining the causes and effects of a failure, while FANSYS provides natural language and the context of a case from which explanations can be based.

9.3 Integration of Memory Structures

Another important area for future work is a closer integration of the knowledge structures used during parsing and those generated in the GI hierarchy through the generalization process. Both hierarchies encode the similar knowledge, but in a different fashion, resulting in redundancy within the system. A more efficient approach would integrate the two memory systems into a single hierarchy which is self-organizing and also capable of supporting the parsing, generalization, and retrieval processes.

9.4 Implementation Scope

One of the next steps for future work is to extend the scope of the project so that more cases can be parsed and understood. This will involve additional work in knowledge representation to encode the cases for the console and cupola. Further work in knowledge representation will also be necessary if a finer granularity of understanding is desired. Continued work on defining additional lexical patterns, inference strategies, and processing MOPs is necessary to handle additional textual cases. Currently, the system encodes approximately 120 lexical patterns and a few domain specific processing strategies, characterizing the parts of the domain necessary to understanding the first ring concentrator case. More lexical knowledge must be added to the system to parse the remaining cases. Question answering sessions could also be made more general and less constrained by defining additional patterns to handle natural language queries from a user of the system. Additionally, the generator would benefit from a more elaborated syntactical system for specifying patterns and from better strategies for choosing which patterns to use during generation. Although lexical and processing knowledge has been encoded for the first case alone, the mechanisms developed for the first case should prove general enough to make the parsing of additional cases involve little more than elaboration of lexical and processing knowledge.

Performance improvements are also possible within FANSYS if its code is optimized. Throughout the implementation, little effort has been made to generate code which is efficient in both runtime and memory requirements. The speed of execution during parsing could be improved by integrating some of the memory search and retrieval techniques. More intelligent elaboration strategies will also improve the speed of execution during memory retrieval. In particular, the last resort strategy of depth-first search should be replaced with more intelligent elaboration strategies. These strategies would require knowledge of the domain to search alternate areas or concepts within memory that apply to the original query. By integrating the memory search functionality with the parser, more efficient access and indexing of MOPs during parsing would be facilitated, speeding up the performance of the parser substantially. Finally, the use of a parallel machine would result in a dramatic increase in memory creation and retrieval speed since many of the traversal paths could then be examined simultaneously.

9.5 Knowledge Acquisition

Self-organization and learning are also extensions that need to be addressed in FANSYS. Although the GI hierarchy is self-organizing during the creation processes, the processes that govern memory retrieval are fixed. A significant improvement to FANSYS would include the capability to learn elaboration strategies as the system is used. As questions are posed and cases retrieved, the system can be trained as to which cases and procedures relate to each other. When presented with a general question or an unknown question, FANSYS could rely upon previous cases for elaboration strategies which may be appropriate.

FANSYS is currently able to read and build the conceptual representation of input text that describes failure diagnosis and repair in a given flight system. However, FANSYS should also be able to build a coherent model of (1) the causal relationships that exists among components of the flight system, (2) the effects and causes of failures in that system, and (3) the procedures used to deal with those failures. This domain model must be dynamically built as a result of

remindings (Schank, 1982) that occur when similar textual inputs are processed. FANSYS should also account for how cross-contextual remindings occur when reading input descriptions of failure analysis in different flight systems, and how memory search and retrieval processes must be able to adapt to memory changes that result from these remindings. To model this reminding process, FANSYS must organize and maintain failure-analysis memories by using a hierarchical organization scheme of the type used the OCCAM system (Pazzani, 1988) in addition to the GI hierarchy.

FANSYS must also be able to handle input text that is not directly dependent on the knowledge constructs and lexical entries initially encoded in the system. To deal with this knowledge engineering bottleneck, FANSYS must be able to dynamically acquire and augment its knowledge during text comprehension. Consequently, the model of text comprehension and question answering implemented in FANSYS will include explanation-based and case-based strategies, such as those described in (Kolodner, 1988) and (Schank and Riesbeck, 1989), for dynamically acquiring and encoding knowledge constructs associated with a given domain or a given natural language.

9.6 Model and Functional Based Reasoning

Another extension which would increase the power of FANSYS is the addition of model-based and functional-based reasoning (Hodges, 1989). These techniques allow for an entirely new level of analysis. In model-based reasoning, the system creates its own model prototype based on the input data. For example, if the system were given that "component A is connected to component B via the network", then an internal model of two connected components via a network would be created. Consequently, the system can then draw inferences and conclusions based on the model. This type of reasoning would be necessary if the user wanted to know what would happen if component A were switched with component B. In this case, the system could switch the components in its model and discover the ensuing results. Similarly, in functional-based reasoning, conclusions are drawn based upon the functions of various devices. Providing FANSYS with both types of reasoning will require the development of new memory structures which accommodate these and the reasoning approaches previously discussed.

9.7 Subjective Comprehension of Input Text and Questions

Comprehension of input text and questions in FANSYS must also be influenced by the *ideological perspective* (Carbonell, 1981) that FANSYS may have about diagnosis and repair procedures in a given domain. That is, FANSYS must attempt to understand descriptions of failure analysis and relate their conceptual content to its own beliefs and justifications involving related and/or similar failures. If FANSYS reads descriptions that are inconsistent with its beliefs, then FANSYS must be able to generate counterarguments as it reads the input text or questions. In order to account for this process of subjective comprehension, FANSYS must have its own ideology, as well as strategies for determining inconsistencies between beliefs in long-term memory and input text, and strategies for selecting counterarguments. These counterargument strategies should be based on the argument-planning knowledge underlying the taxonomy of argument units proposed by Alvarado (1990).

9.8 Belief Inferences Based on Past Failure Analysis

FANSYS must be able to answer questions about causes and effects of failures, and about procedures for repairing failures. Answers to those questions must be found by accessing indices into FANSYS's long-term memory and then traversing memory links associated with those indices. There are questions, however, that require beliefs to be inferred from beliefs and justifications already existing in memory. For example, consider a hypothetical question of the type: "Would FANSYS agree/disagree with advise seeker A over the use of repair procedure P when dealing with failure F?" Here, FANSYS must generate plausibly held beliefs, given its beliefs and justifications in possibly unrelated failure situations. That is, FANSYS must be able to use memories from a previous failure situation to aid in understanding how a repair procedure might be used in a new failure situation. Getting FANSYS to handle hypothetical questions will require modeling the process of belief inference in relation to models of ideology and long-term memory organization.

10. Conclusions

This technical report has described the design and implementation of FANSYS, a failure-analysis system capable of reading segments of manuals dealing with failures on the data management system (DMS) of NASA's Space Station Freedom, and answering questions regarding such failures. FANSYS includes a case-based parser and a model of memory organization, search, and retrieval that are used to create a library of failure-analysis cases. This library is later accessed by FANSYS when answering questions about failures that may occur in the DMS. A major benefit derived from this work has been the formulation of a computational framework for (1) integrating domain-dependent knowledge with case-based parsing strategies within text comprehension systems, (2) representing and indexing failure-analysis cases, (3) creating generalizations among these cases, and (4) using search and retrieval strategies to answer questions about failure causes and effects, as well as failure detection and correction procedures. This computational framework may provide a foundation for developing sophisticated advice-giving and decision-making systems that help operate and maintain NASA's spacecraft.

The experience of designing and implementing a model of text comprehension and question answering for failure analysis has shed light on some of the basic problems any intelligent computer system must address: knowledge representation, organization, application, and retrieval. The model has helped increase understanding of representational constructs and processing strategies needed to analyze system failures. As such, this research has been a step towards modeling basic components that must be present in natural language processing systems.

References

- Alvarado, S. J. (1990). *Understanding Editorial Text: A Computer Model of Argument Comprehension*. Boston, MA: Kluwer Academic Publishers.
- Alvarado, S. J. (1992). Argument Comprehension. In S. C. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence* (Second edition). New York: Wiley.
- Bartlett, R. Davis, G. Grant, T. Gibson, J. Hedges, R. Johnson, M. Liu, Y. Patterson-Hine, A. Sliwa, N. Sowizral, H. and Yan, J. (1992). "Space Station Freedom Data Management System Growth and Evolution Report," *NASA Technical Memorandum 103869*. Ames Research Center.
- Blattner, M. M. and Dannenberg, R. B. (1992). *Multimedia Interface Design*. New York, NY: ACM Press.
- Carbonell, J. G. (1981). *Subjective Understanding: Computer Models of Belief Systems*. Ann Arbor, MI: UMI Research Press.
- Charniak, E. (1977). "A framed PAINTING: The Representation of a Commonsense Knowledge Fragment," *Cognitive Science*, 1, 355-394.
- Charniak, E. (1978). "On the Use of Framed Knowledge in Language Comprehension," *Artificial Intelligence*, 11, 225-265.
- de Kleer, J. and Williams, B. (1987). "Diagnosing Multiple Faults," *Artificial Intelligence*. v32 no. 1, 97-130.
- Dyer, M. G. (1983). *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension*. Cambridge, MA: MIT Press.
- Dyer, M. G. and Lehnert, W. G. (1982). Question Answering for Narrative Memory. In J. F. Le Ny and W. Kintsch (Eds.), *Language and Comprehension*. Amsterdam. North-Holland.
- Flowers, M., McGuire, R., and Birnbaum, L. (1982). Adversary Arguments and the Logic of Personal Attacks. In W. G. Lehnert and M. G. Ringle (Eds.), *Strategies for Natural Language Processing*. Hillsdale, NJ: Lawrence Erlbaum.
- Gentner, D. and Grudin, J. (1990). "Why Good Engineers (Sometimes) Create Bad Interfaces," *Proceedings of CHI 90*, ACM Press, 277-282.
- Hammond, K., and Hurwitz, N. (1988). "Extracting Diagnostic Features from Explanations," *Proceedings of the Workshop on Case-Based Reasoning (DARPA)*. San Mateo, CA: Morgan-Kaufmann Publishers.
- Hendler, J. A. (1988). *Integrating Marker-Passing and Problem-Solving: A Spreading Activation Approach to Improved Choice in Planning*. Hillsdale, NJ: Lawrence Erlbaum.
- Hodges, J. (1989). "Device Representation for Modeling Improvisation in Mechanical Use Situations," *Proc. 11th Ann Meeting Cognitive Science Soc.* Hillsdale, NJ: Lawrence Erlbaum.

- International Business Machines Corporation (July, 1990). "Space Station Program Data Management System," *89IBMX0367R1*, (DR SA-25.1I), Houston, TX.
- Iverson, D. L. and Patterson-Hine, F. A. (1990). Object-Oriented Fault Tree Models Applied to System Diagnosis. *Proceedings of the SPIE Applications of Artificial Intelligence VIII Conference*. Orlando, Florida.
- Kolodner, J. (1983a). "Maintaining Organization in a Dynamic Long-Term Memory," *Cognitive Science*, 7, 243-280.
- Kolodner, J. (1983b). "Reconstructive Memory: A Computer Model," *Cognitive Science*, 7, 281-328.
- Kolodner, J. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*. Hillsdale, NJ: Lawrence Erlbaum.
- Kolodner, J. (Ed.) (1988). *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann. Lebowitz, M. (1988). "The use of Memory in Text Processing," *Communications of the ACM*, V31 N12, 48-62.
- Lehnert, W. G. (1978). *The Process of Question Answering: A Computer Simulation of Cognition*. Hillsdale, NJ: Lawrence Earlbaum.
- Liu, H., Tan, A., Lim, J., & Teh, H. (1991). "Practical Application of a Connectionist Expert System --- The INSIDE Story," *World Congress on Expert Systems*, Orlando, Florida: Pergamon Press, 1-21.
- McDonnell Douglas Space Systems Company. (March, 1990). "Failure Modes and Effects Analysis (FMEA) for the Data Management System (DMS)," *MDC H4563A*, (DR SA-06.1), Huntington Beach, CA.
- McDonnell Douglas Space Systems Company. (July, 1990). "Integrated Operations Requirements For Distributed Systems," *MDC H4380*, (DR SY-46.1), Volume 1, Huntington Beach, CA.
- MacLean, A., Bellotti, V., Young, R. and Moran, T. (1991). "Reaching through Analogy: A Design Rationale Perspective on Roles of Analogy," *Proceedings of CHI 91*, ACM Press, 167-172.
- Minksy, M. (1975). "Framework for Representing Knowledge," In P. Winston (Ed.), *Psychology of Computer Vision*. New York: McGraw-Hill.
- Minksy, M. (1977). "Frame-System Theory," In P. Johnson and O. Wason (Eds.), *Thinking: Readings in Cognitive Science*. Cambridge, MA: MIT Press.
- Myers, B. (1991). "Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs," *Proceedings of UIST '91*, 211-220.
- Norvig, P. (1989). "Marker Passing as a Weak Method for Text Inferencing," *Cognitive Science*, 13, 569-620.

- Pazzani, M. J. (1988). *Learning Causal Relationships: An Integration of Empirical and Explanation-Based Learning Methods* (Ph.D. Dissertation). Computer Science Department. University of California, Los Angeles.
- Patterson-Hine, F. and Iverson, D. (1990). "An Integrated Approach to System Design, Reliability, and Diagnosis," *NASA Technical Memorandum 102861*. Ames Research Center.
- Reed, N. & Johnson, P. (1990). "Generative Knowledge for Computer Troubleshooting," *European Conference on Artificial Intelligence '90*, 535-540.
- Riesbeck, C. K. and Martin, C. E. (1986). "Direct Memory Access Parsing." In. Kolodner, J. and Riesbeck, R. (Eds.), *Experience, Memory, and Reasoning*. Hillsdale, NJ: Lawrence Erlbaum.
- Riesbeck, C. & Schank, R. (1989). *Inside Case-based Reasoning*. Hillsdale, NJ: Lawrence Erlbaum.
- Schank, R. C. (Ed.) (1975). *Conceptual Information Processing*. Amsterdam: North-Holland.
- Schank, R. C. and Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding*. Hillsdale, NJ: Lawrence Erlbaum.
- Schank, R. C. (1978). "What Makes Something 'Ad Hoc.'" *Proceedings of Theoretical Issues in Natural Language Processing-2*. University of Illinois, Urbana-Champaign, pp. 8-13.
- Schank, R. C. and Carbonell, J. G. (1979). "Re: The Gettysburg Address, Representing Social and Political Acts." In N. Findler (Ed.), *Associative Networks*. New York: Academic Press.
- Schank, R. C. (1982). *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge: Cambridge University Press.
- Stevenson, R., Miller, J., and Austin, M. (1991). "Failure Environment Analysis Tool (FEAT) Development Status," *Proceedings of AIAA '91*, 676-682.
- Struss, P. (1988). "Extensions to ATMS-based Diagnosis," In Gero (Ed.), *Artificial Intelligence Engineering: Diagnosis and Learning*. Amsterdam: Elsevier, 3-28.
- Tulving, E. (1972). "Episodic and semantic memory," in E. Tulving & W. Donaldson (Eds.), *Organization of memory*, New York, NY: Academic Press.
- Wilensky, R. (1986). "Knowledge Representation - A Critique and a Proposal." In. Kolodner, J. and Riesbeck, R. (Eds.), *Experience, Memory, and Reasoning*. Hillsdale, NJ: Lawrence Erlbaum.

Appendix A : Case Descriptions

Contents of Case Descriptions

- 1. Ring Concentrator Cases**
- 2. Gateway Cases**
- 3. Standard Data Processor Cases**
- 4. Time Generation Unit Cases**
- 5. Fixed Multi-Purpose Applications Console Cases**
- 6. Mass Storage Unit Cases**
- 7. Cupola Multi-Purpose Applications Console Cases**

* FANSYS : A Computer Model of Text Comprehension and Question
* Answering for Failure Analysis

* File: case_descriptions

* Developed by: Sergio J. Alvarado
* Ronald K. Braun
Kenrick J. Mock

* Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616

* Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

This file contains the case descriptions that were excerpted from the FMEA manuals by McDonnell Douglas (March, 1990). Following each case entry is the hand-coded representation of that case used by the system during question answering sessions.

RING CONCENTRATOR CASES

RC.1

ITEM NAME : Ring Concentrator (RC)

FAILURE MODE : Loss of output - Failure to Start

FAILURE CAUSES : Piece-part failures , Contamination , Temperature (High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Indication of a RC " failure to start " is first detected by the " next " active node on the network . Local System Management (SM) within the " next " node will reach a time-out limit for receipt of the network token .

CORRECTIVE ACTION :

(A) Short Term : Network reconfiguration is effected automatically . The DMS network remains in operation in a reconfigured state with the failed RC bypassed .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . The conditions associated with the replacement of a RC within the network configuration are such that the network is already in a reconfigured state supplying full services .

(B) Mission Support : None .

(C) System : None .

(D) Interfaces : None .

(defmop 1-m-rc-1 (m-case) instance
(failed-component m-generic-RC instance FAILED-NODE)
(mode m-startup-no-output instance
(failed-component m-RC FAILED-NODE)
(detection-node m-next-RC instance DETECTION-NODE)
(cause m-faulty-component-cause instance
(failed-component m-RC FAILED-NODE)
(cause m-contamination-cause instance
(failed-component m-RC FAILED-NODE)
(cause m-temperature-out-of-range-cause instance
(failed-component m-RC FAILED-NODE)
(cause m-mechanical-shock-cause instance
(failed-component m-RC FAILED-NODE)
(cause m-thermal-shock-cause instance
(failed-component m-RC FAILED-NODE)
(detection m-next-node-detection instance
(failed-component m-RC FAILED-NODE)
(detection-component m-rc DETECTION-NODE)
(message m-token instance)
(CORRECTION-PLAN)
(correction m-non-operational-replacement instance
(failed-component m-RC FAILED-NODE)
(correction-component m-crew instance)
(configuration m-core-network instance NETWORK)
(framework m-short-term instance)
(CORRECTION-PLAN)

RC.2

ITEM NAME : Ring Concentrator (RC)

FAILURE MODE : Loss of output - Failure During Operation

FAILURE CAUSES : Piece-part failures , Contamination , Temperature (High or Low) , Mechanical shock , Thermal Shock

(A) Crew / SSPE : None . The conditions associated with the replacement of a RC within the network configuration are such that the network is already in a reconfigured state supplying full services .

FAILURE DETECTION / VERIFICATION : Indication of a RC
 " loss of output " during operations is first detected by
 the " next " active node on the network . Local System Management
 (SM) within the " next " node will reach a time-out limit for receipt
 of the network token .

CORRECTIVE ACTION :
 (A) Short Term : OMS will automatically reconfigure the network and
 present information regarding the reconfigured network to crew
 and ground controllers . Corrective action options consist of (1)
 selector over of all communications / support to the backup network ,
 or (2) by-pass of only the failed RC on the primary network using the
 backup network .

(B) Long Term : Failed RC ORUs are removed and replaced via a logistics
 spare .

FAILURE EFFECT ON :
 (A) CREW / SSPE : None . The conditions associated with the replacement
 of a RC within the network configuration are such that the network is
 already in a reconfigured state supplying full services .

(B) Mission Support : None .

(C) System : None .

(D) Interfaces : None .

ITEM NAME : Ring Concentrator (RC)
FAILURE MODE : Erroneous Output (Transmitter Stuck High / Low) -
Failure During Operation

FAILURE CAUSES : Piece-part failures , Contamination , Temperature
 (High or Low) , Mechanical Shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Indication of a RC stuck at high /
 low failure is first detected by the " next " active node on the
 network . Local System Management (SM) within the " next " node will
 reach a time-out limit for receipt of the network token .

CORRECTIVE ACTION :

(A) Short Term : OMS will automatically reconfigure the network and
 present information regarding the reconfigured network to crew and
 ground controllers . Corrective action options consist of (1)
 selector over of all communications / support to the backup network ,
 or (2) by-pass of only the failed RC on the primary network using
 the backup network .

(B) Long Term : Failed RC ORUs are removed and replaced via a
 logistics spare .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . The conditions associated with the replacement
 of a RC within the network configuration are such that the network
 is already in a reconfigured state supplying full services .

(B) Mission Support : None .

(C) System : None .

(D) Interfaces : None .

(defmop 1-m-rc.3 (m-case) instance
 (failed-component m-generic-RC instance FAILED-NODE)
 (mode m-operational-no-output instance)
 (failed-component m-RC FAILED-NODE)
 (detection-node m-next-RC instance DETECTION-NODE)
 (message m-token instance)
 (cause m-faulty-component-cause instance
 (failed-component m-RC FAILED-NODE))
 (cause m-contamination-cause instance
 (failed-component m-RC FAILED-NODE))
 (cause m-temperature-out-of-range-cause instance
 (failed-component m-RC FAILED-NODE))
 (cause m-mechanical-shock-cause instance
 (failed-component m-RC FAILED-NODE))
 (cause m-thermal-shock-cause instance
 (failed-component m-RC FAILED-NODE))
 (detection m-next-node-detection instance
 (failed-component m-RC FAILED-NODE)
 (detection m-rc DETECTION-NODE)
 (message m-token instance))
 (correction m-switch-or-bypass instance
 (failed-component m-RC FAILED-NODE))
 (correction-component m-sm instance)
 (configuration m-core-network instance NETWORK)
 (backup-configuration m-core-network-backup instance)
 (backup-component m-backup-rc instance)
 (system m-DNS instance)
 (framework m-short-term instance)

CORRECTION PLAN
 (correction m-replace-with-spare instance
 (failed-component m-RC FAILED-NODE))
 (correction-component m-crew instance)

```

(detection-node m-next-rc instance DETECTION-NODE)
  (message m-erroneous-message instance)
  (cause m-faulty-component -cause instance
    (failed-component m-RC FAILED-NODE))
  (cause m-contamination -cause instance
    (failed-component m-RC FAILED-NODE))
  (cause m-temperature-out-of-range -cause instance
    (failed-component m-RC FAILED-NODE))
  (cause m-mechanical-shock -cause instance
    (failed-component m-RC FAILED-NODE))
  (cause m-thermal-shock -cause instance
    (failed-component m-RC FAILED-NODE))
  (detection m-next-node-detection instance
    (failed-component m-RC FAILED-NODE))
  (configuration-component m-rc DETECTION-NODE)
  (message m-token instance) )

  (correction m-switch-or-bypass instance
    (failed-component m-RC FAILED-NODE))
  (correction-component m-smi instance)
  (configuration m-core-network instance NETWORK)
  (backup-configuration m-core-network-backup instance)
  (backup-component m-backup-rc instance)
  (system m-DMS instance)
  (framework m-short-term instance)

CORRECTION-PLAN
  (failed-component m-RC FAILED-NODE)
  (correction-component m-crew instance)
  (configuration m-core-network NETWORK)
  (backup-component m-backup-rc instance)
  (framework m-long-term instance)
  (effect m-failure-effect instance
    (sequence-of-steps m-procedure CORRECTION-PLAN)
    (effected-component m-crew instance))
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effected-component m-mission-support instance))
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effected-component m-systems instance))
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effected-component m-interfaces instance)))
*****
```

GATEWAY CASES

GW.1

```

ITEM NAME : Gateway ( GW )

FAILURE MODE : Loss of output - Failure to Start

FAILURE CAUSES : Piece-part failures , Contamination , Temperature
( High or Low ) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Detection / Verification of failure
to start will occur through the use of a Power On Self Test ( POST ) .

CORRECTIVE ACTION :
```

```

  (A ) Short Term : OMA / SM software directs PMAD to power-off the
    failed GW . A look-up table maintained within DMS selects the redundant
    GW . OMA / SM then directs PMAD to power-on the replacement GW .

  (B ) Long Term : Failed GWs are removed ( TVA ) and replaced via
    transport to / from the ground .

  FAILURE EFFECT ON :
    ( A ) Crew / SSPE : None . Basic station and crew support functions are
      independent of the functioning of the GWs .
    ( B ) Mission Support : None . The JEM network can function autonomously
      as can the ESA network and the DMS network .
    ( C ) System : None . The operation of the DMS network is independent
      of the operation of the GWs .
    ( D ) Interfaces : None . Selectover to the redundant GW is effected .

  (defmop i-m-gw.1 (m-case) instance
    (failed-component m-generic-GW Instance FAILED-NODE)
    (mode m-startup-no-output instance
      (failed-component m-generic-GW FAILED-NODE)
      (detect ion-node m-generic-GW instance DETECTION-NODE)
      (message m-generic-message instance)
      (cause m-faulty-component -cause instance
        (failed-component m-generic-GW FAILED-NODE))
      (cause m-contamination -cause instance
        (cause m-contamination m-generic-GW FAILED-NODE))
      (failed-component m-generic-GW FAILED-NODE)
      (cause m-temperature-out-of-range -cause instance
        (failed-component m-generic-GW FAILED-NODE))
      (cause m-mechanical-shock -cause instance
        (cause m-mechanical-shock m-generic-GW FAILED-NODE))
      (failed-component m-generic-GW FAILED-NODE)
      (cause m-thermal-shock -cause instance
        (failed-component m-generic-GW FAILED-NODE))
      (detection m-POST-detection instance
        (failed-component m-generic-GW FAILED-NODE))
      (failed-component m-generic-GW FAILED-NODE)
      (detaction-component m-generic-GW FAILED-NODE)
      (correction m-lookup-backup instance
        (correct m-generic-GW FAILED-NODE)
        (failed-component m-generic-GW FAILED-NODE)
        (backup-component m-backup-GW instance)
        (configuration m-core-network instance NETWORK)
        (framework m-short-term instance)
        CORRECTION-PLAN)
        (correction m-replace-with-spare instance
          (failed-component m-generic-GW FAILED-NODE)
          (correction-component m-crew instance)
          (configuration m-core-network NETWORK)
          (backup-component m-backup-GW instance)
          (framework m-long-term instance)
          (frame work m-long-term instance)
          (effect m-failure-effect instance
            (sequence-of-steps 1-m-null-procedure)
            (effected-component m-mission-support instance))
          (effect m-failure-effect instance
            (sequence-of-steps 1-m-null-procedure)
            (effected-component m-systems instance))
          (effect m-failure-effect instance
            (sequence-of-steps 1-m-null-procedure)
            (effected-component m-interfaces instance)))
        *****
```

(sequence-of-steps 1-m-null-procedure)
(effected-component m-interfaces instance))

GW.2

ITEM NAME : Gateway (GW)

FAILURE MODE : Loss of output - Failure during operation

FAILURE CAUSES : Piece-part failures , Contamination , Temperature
(High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Failure detection and verification of the GW loss of output will occur via a node health and status data management function of the DMS System Management (SM) . A subtask of this function is the sending of periodic " heartbeat messages " to each ORU attached to the DMS Ring Concentrators (RCs) to verify its operational status . Failure of any of these ORUs to respond to the heartbeat message is indication of an ORU loss of power output failure .

CORRECTIVE ACTION :

(A) Short Term : OMA / SM software directs PMAD to power-off the failed GW . A look-up table maintained within DMS selects the redundant GW . OMA / SM then directs PMAD to power-on the replacement GW .

(B) Long Term : Failed GWs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . Basic station and crew support functions are independent of the functioning of the GWs .

(B) Mission Support : None . The payload network can function autonomously as can the core network .

(C) System : None . The operation of the DMS network is independent of the operation of the GWs .

(D) Interfaces : None . Selectover to the redundant GW is effected .

(defmop 1-m-gw.2 (m-case) Instance
(failed-component m-generic-gw instance FAILED-NODE)
(mode m-operational-no-output instance
(failed-component m-generic-gw FAILED-NODE)
(detection-node m-SM instance DETECTION-NODE)
(message m-generic-message instance)
(cause m-faulty-component-cause instance
(failed-component m-generic-gw FAILED-NODE)
(cause m-contamination-cause instance
(failed-component m-generic-gw FAILED-NODE)
(cause m-temperature-out-of-range-cause instance
(failed-component m-generic-gw FAILED-NODE)
(cause m-mechanical-shock-cause instance
(failed-component m-generic-gw FAILED-NODE)
(cause m-thermal-shock-cause instance
(failed-component m-generic-gw FAILED-NODE))

(detection m-heartbeat-detection instance
(failed-component m-generic-gw FAILED-NODE)
(detection-component m-SM DETECTION-NODE))
(correction m-lookup-backup instance
(failed-component m-generic-gw FAILED-NODE)
(correction-component m-SM instance
(configuration m-backup-GW instance)
(configuration m-core-network instance NETWORK)
(framework m-short-term instance)
CORRECTION PLAN:
(correction m-replace-with-spare instance
(failed-component m-generic-gw FAILED-NODE)
(correction-component m-crew instance)
(configuration m-core-network instance)
(backup-component m-backup-GW instance)
(framework m-long-term instance))
FAILURE EFFECT INSTANCE:
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-crew instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-backup-GW instance)
(framework m-long-term instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-crew instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-systems instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-interfaces instance))

GW.3

ITEM NAME : Gateway (GW)

FAILURE MODE : Erroneous NIA Output (DMS side of the interface) - Failure during operation

FAILURE CAUSES : Piece-part failures , Erroneous input

FAILURE DETECTION / VERIFICATION : Data packet transmission across the DMS network will include Error Correction Code (ECC) . Inclusion of ECC will allow nodes receiving data packets to perform single bit error detection and correction , and two bit error detection .

CORRECTIVE ACTION :

(A) Short Term : OMA / SM software directs PMAD to power-off the failed GW . A look-up table maintained within DMS selects the redundant GW . OMA / SM then directs PMAD to power-on the replacement GW .

(B) Long Term : Failed GWs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . The probability of an output device responding to an erroneous output is considered negligible .

(B) Mission Support : None . The payload network can function autonomously as can the core network .

CASE - ASSUMPTIONS

(C) System : The operation of the DMS network is independent of the operation of the GWs .

(D) Interfaces : None . Selectover to the redundant GW is effected .

```
(defmop 1-m-gw-3 (m-case) instance
  (failed-component m-generic-GW instance FAILED-NODE)
  (mode m-operational-errorneous-output instance)
  (failed-component m-generic-gw FAILED-NODE)
  (detection-node m-SM instance DETECTION-NODE)
  (message m-generic-message instance)
  (cause m-faulty-component-cause instance
    (failed-component m-generic-gw FAILED-NODE))
  (cause m-erroneous-input instance
    (failed-component m-generic-gw FAILED-NODE))
  (detection m-ECC-detection instance)
  (failed-component m-generic-gw FAILED-NODE)
  (detection-component m-SM DETECTION-NODE)
  (correction m-lookup-backup instance)
  (failed-component m-lookup-backup FAILED-NODE)
  (correction m-lookup-GW instance)
  (backup-component m-backup-GW instance)
  (configuration m-core-network instance NETWORK)
  (framework m-short-term instance)
  CORRECTION-PLAN)
  (failed-component m-replace-with-spare instance
    (correction-component m-generic-gw FAILED-NODE)
    (correction-component m-crew instance)
    (configuration m-core-network instance)
    (backup-component m-backup-GW instance)
    (framework m-long-term instance)
    (effect m-failure-effect instance
      (sequence-of-steps 1-m-null-procedure)
      (effect-component m-crew instance)))
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effect-component m-systems instance))
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effect-component m-mission-support instance))
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effect-component m-backup FAILED-NODE)
    (effect m-failure-effect instance
      (sequence-of-steps 1-m-null-procedure)
      (effect-component m-interfaces instance)))
  *****
```

STANDARD DATA PROCESSOR CASES

SDP.1

ITEM NAME : Standard Data Processor (SDP-4B)

FAILURE MODE : Loss of output - Failure to start

FAILURE CAUSES : Piece-part failures , Contamination , Temperature (High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Detection /

Verification of failure to start will occur through the use of a Power On Self Test (POST) .

CORRECTIVE ACTION :

(A) Short Term : OMA / SM software directs PMAD to power-off the failed processor . A look-up table maintained within DMS selects a suitable replacement processor . OMA / SM then directs PMAD to power-on a replacement processor .

(B) Long Term : Failed SDP-4B ORUs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) CREW / SSPE : None . Selectover to a redundant SDP-4B occurs for any application requiring it . In the unlikely event of successive failures of redundant SDP-IBs , basic station and crew support functions can be maintained by crew intervention .

(B) Mission Support : None . Selectover to a redundant SDP-4B occurs for any application requiring it .

(C) System : None . In general the SDP-4B Processors support applications in pairs with the first unit active and the second in a cold backup status .

(D) Interfaces : None . Selectover to the redundant SDP-4B occurs for any application requiring it .

(defmop 1-m-SDP.1 (m-case) instance
 (failed-component m-generic-SP Instance FAILED-NODE)
 (mode m-startup-no-output instance)
 (failed-component m-SDP FAILED-NODE)
 (detection-node m-SDP Instance DETECTION-NODE)
 (message m-generic-c-message instance)
 (cause m-faulty-component-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-connection-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-temperature-out-of-range-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-mechanical-shock-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-thermal-shock-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-short-term instance)
 (detection m-POST-detection instance
 (failed-component m-SDP SUSPECT-NODE))
 (detection m-SDP DETECTION-NODE)
 (correction m-lookup-backup instance)
 (correction-component m-SM instance)
 (failed-component m-SDP FAILED-NODE)
 (backup-component m-backup-SDP instance)
 (configuration m-core-network instance)
 (framework m-short-term instance)
 CORRECTION-PLAN)
 (correction m-replace-with-spare instance
 (failed-component m-SDP FAILED-NODE))
 (correction-component m-crew instance)
 (configuration m-core-network instance)
 (framework m-long-term instance)
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-null-procedure)
 (effect-component m-crew instance)))
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-null-procedure)
 (effect-component m-systems instance))
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-null-procedure)
 (effect-component m-interfaces instance)))

(effected-component m-crew instance)
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-nul-procedure)
 (effected-component m-mission-support instance))
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-nul-procedure)
 (effected-component m-systems instance))
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-nul-procedure)
 (effected-component m-interfaces instance)))

SDP .2

ITEM NAME : Standard Data Processor (SDP-4B)

FAILURE MODE : Loss of output - Failure during operation

(High or Low), Mechanical shock , Contamination , Temperature

FAILURE CAUSES : Piece-part failures , Contamination , Temperature
 FAILURE DETECTION / VERIFICATION : Failure detection and verification
 of the SDP-4B loss of output will occur via a node health and status
 data requested by the DMS System Management software . A subtask of
 this function is the sending of periodic " heartbeat messages " to each
 ORU attached to the DMS Ring Concentrators (RCS) to verify its
 operational status . Failure of any of these ORUs to respond to
 TBD consecutive heartbeat message is indication of an ORU loss of
 power output failure .

CORRECTIVE ACTION :

(A) Short Term : OMA / SM software directs PMAD to power-off the
 failed processor . A look-up table maintained within DMS selects a
 suitable replacement processor . OMA / SM then directs PMAD to power-on
 a replacement processor .

(B) Long Term : Failed SDP-4B ORUs are removed (IVA) and replaced via
 transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . Selectover to a redundant SDP-4B occurs for
 any application requiring it .

(B) Mission Support : None . Selectover to a redundant SDP-4B occurs for
 any application requiring it .

(C) System : None . In general the SDP-4B processors support applications
 in pairs with the first unit active and the second in a cold backup status .

(D) Interfaces : None . Selectover to the redundant SDP-4B occurs for
 any application requiring it .

(demop 1-m-SDP .2 (m-case) instance
 (failed-component m-generic-SDP instance FAILED-NODE)
 (mode m-operational 1-no-output instance
 (failed-component m-SDP FAILED-NODE)
 (detection-node m-SDP 1 instance DETECTION-NODE)
 (message m-generic-message instance))

(cause m-faulty-component-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-contamination-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-temperature-out-of-range-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-mechanical-shock-cause instance
 (failed-component m-SDP FAILED-NODE))
 (cause m-thermal-shock-cause instance
 (failed-component m-SDP FAILED-NODE))
 (detection m-heartbeat-detection instance
 (failed-component m-SDP FAILED-NODE))
 (detection m-SM instance))
 (correction m-lookup-backup instance
 (failed-component m-SDP FAILED-NODE))
 (correction-component m-SM instance)
 (configuration m-core-network instance NETWORK)
 (configuration m-backup-SDP instance)
 (framework m-short-term instance)
 (CORRECTION-PLAN)
 (correction m-replace-with-spare instance
 (failed-component m-SDP FAILED-NODE))
 (correction-component m-crew instance)
 (configuration m-core-network instance)
 (backup-component m-backup-SDP instance)
 (framework m-long-term instance)
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-null-procedure)
 (effected-component m-crew instance))
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-null-procedure)
 (effected-component m-mission-support instance))
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-null-procedure)
 (effected-component m-systems instance))
 (effect m-failure-effect instance
 (sequence-of-steps 1-m-null-procedure)
 (effected-component m-interfaces instance))

SDP .3

ITEM NAME : Standard Data Processor (SDP-4B)

FAILURE MODE : Erroneous Output - BCU / BIA Failure during operation

FAILURE CAUSES : Piece-part failures

FAILURE DETECTION / VERIFICATION : The Bus Control Unit / Bus Interface
 Adapter (BCU / BIA) of the SDP-4B detects transmission errors occurring
 on the Local Bus . The expected undetected bit error rate using a
 single parity bit per word is 10 to the minus 12 power .

CORRECTIVE ACTION :

(A) Short Term : If the SM / OMA recovery process has determined that a
 single bus is no longer properly operational , then the short term
 action may be to continue with support using the capabilities of
 the redundant bus .

If it has been determined by SM / OMA that BCU / BIA have failed and

WaSC_UGENIPUWUS

the SDP-4B is performing high priority functions , short term corrective actions will consist of immediate selectover to a warm backup . If it has been determined by SM / CNA that BCU / BIA have failed and low priority operations are effective , then corrective actions will consist of SDP-4B replacement or reassignment of the resident function to a cold backup .

(B) Long Term : Failed SDP-4B ORUs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . Detected erroneous transmissions will be discarded resulting in no impact to crew / SSPE .

(B) Mission Support : None . Selectover to a redundant SDP-4B occurs for any application requiring it .

(C) System : None . In general the SDP-4B processors support applications in pairs with the first unit active and the second in a cold backup status .

(D) Interfaces : None . Selectover to the redundant SDP-4B occurs for any application requiring it .

```
(defmop l-m-SDP .3 (m-case) Instance
  (failed-component m-generic-SDP instance FAILED-NODE)
  (mode m-operational-erroneous-output instance)
  (failed-component m-SDP FAILED-NODE)
  (detected-node m-SM instance DETECTION-NODE)
  (message m-generic-message instance)
  (cause m-faulty-component-cause instance)
  (failed-component m-SDP FAILED-NODE))
  (detected-component m-undefined-detection)
  (sequence m-undefined-and-group instance)
  (correction m-undefined-correct)
  (failed-component m-adp FAILED-NODE)
  (sequence-of-steps m-undefined-and-group instance)
  (framework m-short-term instance)
  (correction m-replace-with-spare instance)
  (failed-component m-SDP FAILED-NODE)
  (correction-component m-adp FAILED-NODE)
  (configuration m-core-network instance)
  (backup-component m-backup-SDP instance)
  (framework m-long-term instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-crew instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-mission-support instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-systems instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-interfaces instance))
```

TIME GENERATION UNIT CASES

TGU.1

ITEM NAME : Time Generation Unit (TGU)

FAILURE MODE : Loss of output - Failure to Start

FAILURE CAUSES : Piece-part failures , Contamination , Temperature (High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Detection / Verification of failure to start will occur through the use of a Power On Self Test (POST) .

CORRECTIVE ACTION :

(A) Short Term : TBA

(B) Long Term : Failed TGUs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None .

(B) Mission Support : TBA

(C) System : None . The operation of the DMS network is independent of the operation of the TGU .

(D) Interfaces : TBA

```
(defmop l-m-TGU.1 (m-case) Instance
  (failed-component m-generic-TGU instance FAILED-NODE)
  (mode m-startup-no-output instance)
  (failed-component m-TGU FAILED-NODE)
  (detected-node m-TGU instance DETECTION-NODE)
  (message m-generic-message instance)
  (cause m-faulty-component-cause instance)
  (failed-component m-TGU FAILED-NODE)
  (cause m-contamination-cause instance)
  (failed-component m-TGU FAILED-NODE)
  (cause m-temperature-out-of-range-cause instance)
  (failed-component m-TGU FAILED-NODE)
  (cause m-mechanical-shock-cause instance)
  (failed-component m-TGU FAILED-NODE)
  (cause m-thermal-shock-cause instance)
  (failed-component m-TGU FAILED-NODE)
  (detected m-POST-detection instance)
  (failed-component m-TGU SUSPECT-NODE)
  (detected-component m-TGU DETECTION-NODE)
  (correction m-undefined-correct)
  (failed-component m-TGU FAILED-NODE)
  (sequence-of-steps m-undefined-and-group instance)
  (framework m-short-term instance)
  (correction m-replace-with-spare instance)
  (failed-component m-TGU FAILED-NODE)
  (correction-component m-TGU FAILED-NODE)
  (configuration m-core-network instance)
  (framework m-long-term instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-crew instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-mission-support instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-systems instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-interfaces instance))
```

TIME GENERATION UNIT CASES

```

(sequence-of-steps 1-m-null-procedure)
  (effected-component m-crew instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
    (effected-component m-mission-support instance)
    (effect m-failure-effect instance)
    (sequence-of-steps 1-m-null-procedure)
      (effected-component m-systems instance)
    (effect m-failure-effect instance)
    (sequence-of-steps 1-m-null-procedure)
      (effected-component m-interfaces instance))
*****  

TGU.2  

ITEM NAME : Time Generation Unit ( TGU )  

FAILURE MODE : Loss of output - Failure during operation  

FAILURE CAUSES : Piece-part failures , Contamination , Temperature
  ( High or Low ) , Mechanical shock , Thermal Shock  

FAILURE DETECTION / VERIFICATION : Indications of loss of output and
self test capabilities occur via presence of the GPS input signal ,
quality of the GPS input signal , loss of internal clock , quality of
internal clock signal , detection of local bus errors , detection of
memory errors ( parity , ECC ) and Time Distribution Bus ( TBD ) driver
Failure .  

*****  

TGU.3  

CORRECTIVE ACTION :  

  ( A ) Short Term : TBA  

  ( B ) Long Term : Failed TGUs are removed ( IVA ) and replaced via
transport to / from the ground .  

FAILURE EFFECT ON :  

  ( A ) Crew / SSPE : None .  

  ( B ) Mission Support : TBA  

  ( C ) System : None .  

  ( D ) Interfaces : TBA  

*****  

defmop 1-m-TGU.2 (m-case) instance
  (failed-component m-generic-TGU instance FAILED-NODE)
  (mode m-operational-no-output instance
    (failed-component m-TGU FAILED-NODE)
    (detection-node m-TGU instance DETECTION-NODE)
    (message m-generic-message instance)
    (cause m-faulty-component-cause instance
      (failed-component m-TGU FAILED-NODE)
      (cause m-contamination-cause instance
        (failed-component m-TGU FAILED-NODE)
        (cause m-temperature-out-of-range-cause instance
          (failed-component m-TGU FAILED-NODE)
          (cause m-mechanical-shock-cause instance
            (failed-component m-TGU FAILED-NODE)))

```

```

(cause m-thermal-shock-cause instance
  (failed-component m-TGU FAILED-NODE)
  (detection m-undefined-detection
    (failed-component m-sdp FAILED-NODE)
    (sequence-of-steps m-undefined-and-group instance)
    (correction m-undefined-correction
      (failed-component m-mdp FAILED-NODE)
      (sequence-of-steps m-undefined-and-group instance)
      (framework m-short-term instance)
      (replace-with-spare instance
        (failed-component m-TGU FAILED-NODE)
        (correction-component m-crew instance)
        (configuration m-core-network instance)
        (configuration m-backup-TGU instance)
        (framework m-long-term instance)
        (frame m-long-term instance)
        (effect m-failure-effect instance
          (sequence-of-steps 1-m-null-procedure)
          (effected-component m-crew instance)
          (sequence-of-steps 1-m-null-procedure)
          (effected-component m-mission-support instance)
          (effect m-failure-effect instance
            (sequence-of-steps 1-m-null-procedure
              (effected-component m-systems instance)
              (effected-component m-interfaces instance)
              (effect m-failure-effect instance
                (sequence-of-steps 1-m-null-procedure
                  (effected-component m-interfaces instance))))))))))))  

*****  

TGU.3  

ITEM NAME : Time Generation Unit ( TGU )  

FAILURE MODE : Erroneous Output - Failure during operation  

FAILURE CAUSES : Piece-part failures , Erroneous Input  

FAILURE DETECTION / VERIFICATION : To be supplied .  

CORRECTIVE ACTION :  

  ( A ) Short Term : TBA  

  ( B ) Long Term : Failed TGUs are removed ( IVA ) and replaced via
transport to / from the ground .  

FAILURE EFFECT ON :  

  ( A ) Crew / SSPE : None .  

  ( B ) Mission Support : TBA  

  ( C ) System : None .  

  ( D ) Interfaces : TBA  

*****  

defmop 1-m-TGU.3 (m-case) instance
  (failed-component m-generic-TGU instance FAILED-NODE)
  (mode m-operational-erroneous-output instance
    (failed-component m-TGU FAILED-NODE)
    (cause m-case instance
      (failed-component m-TGU FAILED-NODE)
      (cause m-temperature-out-of-range-cause instance
        (failed-component m-TGU FAILED-NODE)
        (cause m-mechanical-shock-cause instance
          (failed-component m-TGU FAILED-NODE)))

```

(detection-node m-SM instance DETECTION-NODE)

(message m-generic-message instance)

(cause m-erroneous-input instance)

(failed-component m-TGU FAILED-NODE)

(cause m-faulty-component-cause instance)

(failed-component m-TGU FAILED-NODE)

(detection m-undefined-detection

(failed-component m-SDP FAILED-NODE)

(sequence-of-steps m-undefined-and-group instance)

(failed-component m-SDP FAILED-NODE)

(sequence-of-steps m-undefined-and-group instance)

(framework m-short-term instance)

(correction-component m-TGU FAILED-NODE)

(correction-component m-crew instance)

(configuration m-core-network instance)

(backup-component m-backup-TGU instance)

(framework m-long-term instance)

(effect m-failure-effect instance)

(effect m-failure-effect instance)

(framework m-short-term instance)

(effect m-failure-effect instance)

(sequence-of-steps 1-m-null-procedure)

(effected-component m-crew instance)

(sequence-of-steps 1-m-null-procedure)

(effected-component m-mission-support instance)

(effect m-failure-effect instance)

(sequence-of-steps 1-m-null-procedure)

(effected-component m-systems instance)

(effect m-failure-effect instance)

(sequence-of-steps 1-m-null-procedure)

(effected-component m-interfaces instance))

FIXED MULTI-PURPOSE APPLICATIONS CONSOLE CASES

FMPAC.1

ITEM NAME : Fixed Multi-Purpose Applications Console (F-MPAC) (Processor)

FAILURE MODE : Loss of output - Failure to Start

FAILURE CAUSES : Piece-part failures , Contamination , Temperature
(High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Detection / Verification of failure
to start will occur through the use of a Power On Self Test (POST).

CORRECTIVE ACTION :

(A) Short Term : OMA / SM software directs PMAD to power-off the failed F-MPAC . A look-up table maintained within DMS selects the replacement F-MPAC . OMA / SM then directs PMAD to power-on the replacement F-MPAC .

(B) Long Term : Failed F-MPACs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . The crew can use a redundant F-MPAC .

(B) Mission Support : None . The crew can use a redundant F-MPAC .

(C) System : None . The operation of the DMS network is independent of the operation of the F-MPAC processors .

(D) Interfaces : None . The crew can use a redundant F-MPAC .

(defmop 1-m-FMPAC.1 (m-case) instance
(failed-component m-generic-FMPAC Instance FAILED-NODE)
(mode m-startup-no-output instance
(failed-component m-FMPAC FAILED-NODE)
(detection-node m-FMPAC Instance DETECTION-NODE)
(message m-generic-c-message instance)
(cause m-faulty-component-cause instance
(cause m-faulty-component-cause instance
(failed-component m-FMPAC FAILED-NODE)
(cause m-contamination-cause instance
(failed-component m-FMPAC FAILED-NODE)
(cause m-temperature-out-of-range-cause instance
(cause m-mechanical-shock-cause instance
(cause m-thermal-shock-cause instance
(cause m-thermal-shock-cause instance
(failed-component m-FMPAC FAILED-NODE)
(detection m-POST-detection instance
(failed-component m-FMPAC SUSPECT-NODE)
(detection-component m-FMPAC DETECTION-NODE)
(correction m-lookup-backup instance
(correction m-backup-FMPAC instance
(correction m-crew instance
(failed-component m-FMPAC FAILED-NODE)
(backup-component m-backup-FMPAC instance)
(framework m-short-term instance)
CORRECTION-PLAN
(correction m-non-operational-replacement instance
(failed-component m-FMPAC FAILED-NODE)
(correction-component m-crew instance)
(configuration m-core-network NETWORK)
(backup-component m-backup-FMPAC instance)
(framework m-long-term instance)
(effect m-failure-effect instance
(sequenced-of-steps 1-m-null-procedure)
(effected-component m-crew instance)

ITEM NAME : Fixed Multi-Purpose Applications Console (F-MPAC) (Processor)

FMPAC.2

ITEM NAME : Fixed Multi-Purpose Applications Console (F-MPAC) (Processor)

FAILURE MODE : Loss of output - Failure during operation

FAILURE CAUSES : Piece-part failures , Contamination , Temperature

(High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Failure detection and verification of the loss of output will occur via a node health and status data management function of the DMS System Management software . A subtask of this function is the sending of Periodic " heartbeat " messages " to each ORU attached to the DMS Ring Concentrators (RCS) to verify its operational status . Failure of any of these nodes to respond to the heartbeat message is indication of a nodal loss of output failure .

CORRECTIVE ACTION :

- (A) Short Term : OMA / SM software directs PMAD to power-off the failed F-MPAC . A look-up table maintained within DMS selects the replacement F-MPAC . OMA / SM then directs PMAD to power-on the replacement F-MPAC .
- (B) Long Term : Failed F-MPACs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

- (A) Crew / SSPE : None . The crew selects a redundant F-MPAC processor .
- (B) Mission Support : None . Selectover to a redundant F-MPAC processor occurs .
- (C) System : None . The operation of the DMS network is independent of the operation of the F-MPAC processors .
- (D) Interfaces : None . Selectover to a redundant F-MPAC occurs .

```
(defmop 1-m-FMPAC.2 (m-case) instance
  (failed-component m-generic-FMPAC instance FAILED-NODE)
  (mode m-operational-no-output instance
    (failed-component m-FMPAC FAILED-NODE)
    (detection-node m-SH instance DETECTION-NODE)
    (message m-generic-message instance))
  (cause m-faulty-component-cause instance
    (failed-component m-FMPAC FAILED-NODE))
  (cause m-contamination-cause instance
    (failed-component m-FMPAC FAILED-NODE))
  (cause m-temperature-out-of-range-cause instance
    (failed-component m-FMPAC FAILED-NODE))
  (cause m-mechanical-shock-cause instance
    (failed-component m-FMPAC FAILED-NODE))
  (cause m-thermal-shock-cause instance
    (failed-component m-FMPAC FAILED-NODE))
  (detection m-heartbeat-detection instance
    (failed-component m-FMPAC FAILED-NODE))
  (detection-component m-SH DETECTION-NODE))
  (correction m-lookup-backup instance
    (failed-component m-FMPAC FAILED-NODE))
  (correction-component m-SM instance)
  (backup-component m-backup-FMPAC instance)
  (framework m-short-term instance))

CORRECTION-PLAN)
  (correction m-non-operational-replacement instance
    (failed-component m-FMPAC FAILED-NODE)
    (correction-component m-SM instance)
    (backup-component m-backup-FMPAC instance)
    (framework m-short-term instance)

  (configuration m-core-network NETWORK)
  (backup-component m-backup-FMPAC instance)
  (failed-component m-FMPAC FAILED-NODE))
```

(framework m-long-term Instance)

```
(effect m-failure-effect instance
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-crew instance)
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effected-component m-mission-support instance))
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effected-component m-systems instance))
  (effect m-failure-effect instance
    (sequence-of-steps 1-m-null-procedure)
    (effected-component m-interfaces instance)))
```

FMPAC.3

ITEM NAME : Fixed Multi-Purpose Applications Console (F-MPAC) (Processor)

FAILURE MODE : Erroneous NIA Output - Failure during operation

FAILURE CAUSES : Piece-part failures

FAILURE DETECTION / VERIFICATION : Data packet transmission across the DMS network will include Error Correction Code (ECC) . Inclusion of ECC will allow nodes receiving data packets to perform single bit error detection and correction , and two bit error detection .

CORRECTIVE ACTION :

- (A) Short Term : Two possible corrective actions are (1) selectover to the backup network , and (2) selectover to a warm or cold backup F-MPAC processor .
 - (B) Long Term : Failed F-MPACs are removed (IVA) and replaced via transport to / from the ground .
- FAILURE EFFECT ON :
- (A) Crew / SSPE : None . SM / OMA will direct the processor to continue operations using the backup network communications .
 - (B) Mission Support : None . Selectover to a redundant F-MPAC processor occurs .
 - (C) System : None . The operation of the DMS network is independent of the operation of the F-MPAC processors .
 - (D) Interfaces : None . The crew can use a redundant F-MPAC .

```
(defmop 1-m-FMPAC.3 (m-case) instance
  (failed-component m-generic-FMPAC instance FAILED-NODE)
  (mode m-operational-errorous-output instance
    (failed-component m-FMPAC FAILED-NODE)
    (detection-node m-SH instance DETECTION-NODE)
    (message m-generic-message instance))
  (cause m-faulty-component-cause instance
    (failed-component m-FMPAC FAILED-NODE))
  (cause m-thermal-shock-cause instance
    (failed-component m-FMPAC FAILED-NODE))
  (detection m-heartbeat-detection instance
    (failed-component m-FMPAC FAILED-NODE))
  (detection-component m-SH DETECTION-NODE))
  (correction m-lookup-backup instance
    (failed-component m-FMPAC FAILED-NODE))
  (correction-component m-SM instance)
  (backup-component m-backup-FMPAC instance)
  (framework m-short-term instance))

CORRECTION-PLAN)
  (correction m-non-operational-replacement instance
    (failed-component m-FMPAC FAILED-NODE)
    (correction-component m-SM instance)
    (backup-component m-backup-FMPAC instance)
    (framework m-short-term instance)

  (configuration m-core-network NETWORK)
  (backup-component m-backup-FMPAC instance)
  (failed-component m-FMPAC FAILED-NODE))
```

(detection-component m-SM DETECTION-NODE)

(correction m-lookup-backup Instance

(failed-component m-FMPAC FAILED-NODE)

(correction-component m-FMPAC Instance)

(backup-component m-backup-FMPAC Instance)

(framework m-short-term instance))

(correction m-non-operational-replacement instance

(failed-component m-FMPAC FAILED-NODE)

(correction-component m-backup-FMPAC Instance)

(configuration m-core-NETWORK)

(backup-component m-backup-FMPAC Instance)

(framework m-long-term instance)

(effect m-failure-effect instance

(sequence-of-steps i-m-null-procedure)

(effected-component m-crew Instance)

(effect m-failure-effect Instance

(sequence-of-steps i-m-null-procedure)

(effected-component m-crew Instance))

(effect m-failure-effect Instance

(sequence-of-steps i-m-null-procedure)

(effected-component m-mission-support Instance))

(effected-component m-mission-support Instance)

(effect m-failure-effect instance

(sequence-of-steps i-m-null-procedure)

(effected-component m-interfaces Instance))

MASS STORAGE UNIT CASES

MSD.1

ITEM NAME : Mass Storage Device (MSD)

FAILURE MODE : Loss of output - Failure to Start

FAILURE CAUSES : Piece-part failures , Contamination , Temperature
 (High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : To be supplied .

CORRECTIVE ACTION :

(A) Short Term : TBA

(B) Long Term : Failed MSDs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . The OMA / SM selects a redundant MSD .

(B) Mission Support : None . Selectover to a redundant MSD occurs for any application requiring it .

(C) System : None .

(D) Interfaces : None . Selectover to a redundant MSD occurs for any application requiring it .

(defmop 1-m-MSD.1 (m-case) instance

(failed-component m-generic-MSD instance FAILED-NODE)

(mode m-startup-no-output Instance

(failed-component m-MSD FAILED-NODE)

(detection-node m-TGU Instance DETECTION-NODE)

(message m-generic-message instance)

(cause m-faulty-component-cause instance

(failed-component m-MSD FAILED-NODE)

(cause m-contamination-cause instance

(failed-component m-MSD FAILED-NODE)

(cause m-temperature-out-of-range-cause instance

(failed-component m-MSD FAILED-NODE)

(cause m-mechanical-shock-cause instance

(failed-component m-MSD FAILED-NODE)

(cause m-thermal-shock-cause instance

(failed-component m-MSD FAILED-NODE)

(correction m-non-operational-replacement instance

(failed-component m-MSD FAILED-NODE)

(correction-component m-crew Instance

(configuration m-core-network NETWORK)

(backup-component m-backup-MSD Instance)

(framework m-long-term instance)

(effect m-failure-effect instance

(sequence-of-steps i-m-null-procedure)

(effected-component m-crew Instance))

(effect m-failure-effect instance

(sequence-of-steps i-m-null-procedure)

(effected-component m-interfaces Instance))

MSD.2

ITEM NAME : Mass Storage Device (MSD)

FAILURE MODE : Loss of output - Failure during operation

FAILURE CAUSES : Piece-part failures , Contamination , Temperature
 (High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : To be supplied .

CORRECTIVE ACTION :

(A) Short Term : TBA

(B) Long Term : Failed MSDs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . The OMA / SM can use a redundant F-MPAC with its MSD if required .

(B) Mission Support : None . The crew can select a redundant F-MPAC with MSD for any application requiring it .

(C) System : None .

{ D) Interfaces : None . Selectover to a redundant MSD occurs for any application requiring it .

(defmop 1-mMSD.2 (m-case) instance
(failed-component m-generic-MSD instance FAILED-NODE)
(mode m-operational-no-output instance
(failed-component m-MSD FAILED-NODE)
(detected-node m-MSD instance DETECTION-NODE)
(message m-generic-message instance)
(cause m-faulty-component-cause instance
(failed-component m-MSD FAILED-NODE))
(cause m-contaminant-cause instance
(failed-component m-MSD FAILED-NODE))
(cause m-temperature-out-of-range-cause instance
(failed-component m-MSD FAILED-NODE))
(cause m-mechanical-shock-cause instance
(failed-component m-MSD FAILED-NODE))
(cause m-thermal-shock-cause instance
(failed-component m-MSD FAILED-NODE))
(correction m-non-operational-replacement instance
(failed-component m-MSD FAILED-NODE))
(correction-component m-crew instance)
(configuration m-core-network NETWORK)
(configuration m-core-network NETWORK)
(backup-component m-backup-MSD instance)
(framework m-long-term instance)
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effect m-failure-effect m-crew instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effect m-failure-effect m-systems instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)

MSD.3

ITEM NAME : Mass Storage Device (MSD)

FAILURE MODE : Erroneous Output - Missing Data During operation

FAILURE CAUSES : Piece-part failures

FAILURE DETECTION / VERIFICATION : To be supplied .

CORRECTIVE ACTION :

(A) Short Term : TBA

(B) Long Term : Failed MSDs are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . The ORA / SM can use a redundant F-MPAC with its

MSD if required .

{ B) Mission support : None . The crew can select a redundant F-MPAC with MSD for any application requiring it .

(C) System : None .

(D) Interfaces : None . The crew selects a redundant F-MPAC with its attached MSD .

(defmop 1-mMSD.3 (m-case) instance
(failed-component m-generic-MSD instance FAILED-NODE)
(mode m-operational-erroneous-output instance
(failed-component m-MSD FAILED-NODE))
(detection-node m-SM instance DETECTION-NODE)
(message m-generic-message instance))
(cause m-faulty-component-cause instance
(failed-component m-MSD FAILED-NODE))
(cause m-thermal-shock-cause instance
(failed-component m-MSD FAILED-NODE))
(cause m-thermal-shock-cause instance
(failed-component m-MSD FAILED-NODE))
(correction m-non-operational-replacement instance
(failed-component m-MSD FAILED-NODE))
(correction-component m-crew instance)
(configuration m-core-network NETWORK)
(configuration m-core-network NETWORK)
(backup-component m-backup-MSD instance)
(framework m-long-term instance)
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effect m-failure-effect m-crew instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effect m-failure-effect m-systems instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)

CUPOLA MULTI-PURPOSE APPLICATIONS CONSOLE

CMPAC.1

ITEM NAME : Cupola Multi-Purpose Applications Console (C-MPAC) Processor

FAILURE MODE : Loss of output - Failure to Start

FAILURE CAUSES : Piece-part failures , Contamination , Temperature (High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Detection / Verification of failure to start will occur through the use of a Power On Self Test (POST) .

CORRECTIVE ACTION :

(A) Short Term : DMS SM and OMA software directs PMAD to power-off the failed C-MPAC processor . The crew can then replace this processor with a logistics spare and resume the startup operation .

(B) Long Term : Failed C-MPAC processors are removed (IVA) and

replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . The crew can selectover to the redundant C-MPAC should one fail to start . In addition , for certain applications that are not limited to hemispherical viewing from the cupola , the crew can select instead a F-MPAC .

(B) Mission Support : None . The crew can selectover to the redundant C-MPAC should one fail to start . In addition , for certain applications that are not limited to hemispherical viewing from the cupola , the crew can select instead a F-MPAC .

(C) System : None . The operation of the DMS network is independent of the operation of the C-MPAC processors .

(D) Interfaces : None . The crew can selectover to the redundant C-MPAC should one fail to start .

No case representation has been developed for CMPAC.1.

CMPAC.2

ITEM NAME : Cupola Multi-Purpose Applications Console (C-MPAC) Processor

FAILURE MODE : Loss of output - Failure during operation

FAILURE CAUSES : Piece-part failures , Contamination , Temperature (High or Low) , Mechanical shock , Thermal Shock

FAILURE DETECTION / VERIFICATION : Failure detection and verification of the loss of output will occur via a node health and status data management function of the DMS System Management (SM) software . A subtask of this function is the sending of periodic " heartbeat messages " to each ORU attached to the DMS Ring Concentrators (RCs) to verify its operational status . Failure of any of these nodes to respond to TBD consecutive heartbeat message is indication of a nodal failure .

CORRECTIVE ACTION :

(A) Short Term : For operations requiring a hemispherical viewing capability , the crew must first determine whether or not these operations require this viewing from the particular cupola housing the failed C-MPAC processor . If viewing from that cupola is essential , then no redundancy exists and replacement of the failed C-MPAC processor must take place before operations can resume . If viewing and operations can be accomplished instead from the other cupola , then appropriate notification is made to all command and control operations personnel , and operations are conducted from the other cupola .

For operations not requiring a hemispherical viewing capability , a look-up table maintained within DMS selects a suitable replacement F-MPAC workstation to perform the function of the C-MPAC workstation . DMS SM then directs PMAD to power-on the replacement processor .

(B) Long Term : Failed C-MPAC processors are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . Basic station and crew support functions are

Independent of the C-MPAC .

(B) Mission Support : For operations requiring a hemispherical viewing capability , the crew must first determine whether or not these operations require this viewing from the particular cupola housing the failed C-MPAC processor . If viewing from that cupola is essential , then no redundancy exists and replacement of the failed C-MPAC processor must take place before operations can resume . If viewing and operations can be accomplished instead from the other cupola , then appropriate notification is made to all command and control operations personnel , and operations are conducted from the other cupola .

For operations not requiring a hemispherical viewing capability , a look-up table maintained within DMS selects a suitable replacement F-MPAC workstation to perform the function of the C-MPAC workstation . DMS SM then directs PMAD to power-on the replacement processor .

(C) System : None . The operation of the DMS network is independent of the operation of the C-MPAC processors .

(D) Interfaces : None . The DMS network continues to function despite the failure to function of the C-MPAC .

No case representation has been developed for CMPAC.2.

CMPAC.3

ITEM NAME : Cupola Multi-Purpose Applications Console (C-MPAC) Processor

FAILURE MODE : Erroneous NIA Output - Failure during operation

FAILURE CAUSES : Piece-part failures

FAILURE DETECTION / VERIFICATION : Data packet transmission across the network will include Error Correction Code (ECC) . Inclusion of ECC will allow nodes receiving data packets to perform single bit error detection and correction , and two bit error detection .

CORRECTIVE ACTION :

(A) Short Term : For operations requiring a hemispherical viewing capability , the crew must first determine whether or not these operations require this viewing from the particular cupola housing the failed C-MPAC processor . If viewing from that cupola is essential , then no redundancy exists and replacement of the failed C-MPAC processor must take place before operations can resume . If viewing and operations can be accomplished instead from the other cupola , then appropriate notification is made to all command and control operations personnel , and operations are conducted from the other cupola .

For operations not requiring a hemispherical viewing capability , a look-up table maintained within DMS selects a suitable replacement F-MPAC workstation to perform the function of the C-MPAC workstation . DMS SM then directs PMAD to power-on the replacement processor .

(B) Long Term : Failed C-MPAC processors are removed (IVA) and replaced via transport to / from the ground .

FAILURE EFFECT ON :

(A) Crew / SSPE : None . Basic station and crew support functions are

FAILURE EFFECT ON :

(A) Crew / SSPE : None . Basic station and crew support functions are independent of the C-MPAC .

(B) Mission Support : For operations requiring a hemispherical viewing capability , the crew must first determine whether or not these operations require this viewing from the particular cupola housing the failed C-MPAC processor . If viewing from that cupola is essential , then no redundancy exists and replacement of the failed C-MPAC processor must take place before operations can resume . If viewing and operations can be accomplished instead from the other cupola , then appropriate notification is made to all command and control operations personnel , and operations are conducted from the other cupola .

For operations not requiring a hemispherical viewing capability , a look-up table maintained within DMS selects a suitable replacement F-MPAC workstation to perform the function of the C-MPAC workstation . DMS SM then directs PMAD to power-on the replacement processor .

(C) System : None . The operation of the DMS network is independent of the operation of the C-MPAC Processors .

(D) Interfaces : None . The DMS network continues to function despite the failure to function of the C-MPAC .

No case representation has been developed for CMFAC.3.

Appendix B : Detailed I/O Examples

Contents of Detailed I/O Examples

1. Parse of RC.1

Full Parse Trace

Expansion of Final Instantiated Case

2. Parse of a Question

3. Generation of a MOP

4. Memory Retrieval Traces

Direct Retrieval Example

Alternate Entry Point Example

Elaboration Example

Last Resort Depth-First Search Example

FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis

File: parse_trace

Developed by: Sergio J. Alvarado
Ronald K. Braun
Kenrick J. Mock

Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616

Funds for the support of this study have been allocated by the NASA-Ames Research Center, Moffett Field, California, under Intercharge No. NCA2-721.

(ITEM NAME *COLON* RING CONCENTRATOR FAILURE MODE *COLON* LOSS OF OUTPUT - FAILURE TO START FAILURE CAUSES *COLON* PIECE-PART FAILURES *COMMA* CONTAMINATION *COMMA* TEMPERATURE *LEFT-PAREN* HIGH OR LOW *RIGHT-PAREN* *COMMA* MECHANICAL SHOCK *COMMA* THERMAL SHOCK FAILURE DETECTION *SLASH* VERIFICATION *COLON* INDICATION OF A RC FAILURE IS FIRST DETECTED BY THE *QUOTE* NEXT *QUOTE* ACTIVE NODE ON THE NETWORK *PERIOD* SYSTEM MANAGEMENT WILL REACH A TIME-OUT LIMIT FOR RECEIPT OF THE NETWORK TOKEN *PERIOD* CORRECTIVE ACTION *COLON* *LEFT-PAREN* A *RIGHT-PAREN* SHORT TERM *COLON* NETWORK RECONFIGURATION IS EFFECTED AUTOMATICALLY *PERIOD* THE DMS NETWORK REMAINS IN OPERATION IN A RECONFIGURED STATE WITH THE FAILED RC BYPASSED *PERIOD* *LEFT-PAREN* B *RIGHT-PAREN* LONG TERM *COLON* THE CREWMEN CHECK FOR *QUOTE* APPLIED POWER *QUOTE* AND THE CREWMEN CHECK FOR *QUOTE* CONNECTOR TIGHTNESS *QUOTE* *PERIOD* IF THE RC CANNOT THEN BE PLACED IN OPERATION *COMMA* THE RC IS REMOVED AND REPLACED WITH AN ORU LOGISTICS SPARE *PERIOD* FAILURE EFFECT ON *COLON* *LEFT-PAREN* A *RIGHT-PAREN* CREW *SLASH* SSB *COLON* NONE *PERIOD* THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD* *LEFT-PAREN* B *RIGHT-PAREN* MISSION SUPPORT *COLON* NONE *PERIOD* *LEFT-PAREN* C *RIGHT-PAREN* SYSTEM *COLON* NONE *PERIOD* *LEFT-PAREN* D *RIGHT-PAREN* INTERFACES *COLON* NONE *PERIOD* *EOC*).

Reading *COLON* - M-ITEM-CONTEXT referenced
(ITEM NAME *COLON*) = M-ITEM-CONTEXT
Recognizing: M-ITEM-CONTEXT
Creating: I-M-ITEM-CONTEXT .418
Specializing: I-M-ITEM-CONTEXT .418
Recognizing: M-CONTEXT
Creating: I-M-CONTEXT .418
Specializing: I-M-CONTEXT .418
Removing: I-M-IP.388.417

Reading RING
Recognizing: M-IP.327
Creating: I-M-IP.327.421
Specializing: I-M-IP.327.421
Recognizing: I-M-TERMINATOR.323
(RING * CONCENTRATOR) = M-GENERIC-RC

Reading CONCENTRATOR
(RING CONCENTRATOR *) = M-GENERIC-RC referenced
Recognizing: M-GENERIC-RC
Creating: I-M-GENERIC-RC.422
Specializing: I-M-GENERIC-RC.422
Recognizing: M-HL-PATTERN.387
Creating: I-M-HL-PATTERN.387.423
Specializing: I-MHL-PATTERN.387.423
Removing: I-M-IP.327.421

Reading FAILURE
Recognizing: M-IP.389
Creating: I-M-IP.389.424
Specializing: I-M-IP.389.424
Recognizing: I-M-TERMINATOR.323
(FAILURE MODE * :COLON*) = M-MODE-CONTEXT

Reading MODE
Recognizing: M-IP.389
Creating: I-M-IP.389.424
Specializing: I-M-IP.389.424
Recognizing: I-M-TERMINATOR.323
(FAILURE MODE * :COLON*) = M-MODE-CONTEXT

Reading *COLON*
(FAILURE MODE *:COLON*) = M-MODE-CONTEXT referenced
Recognizing: M-MODE-CONTEXT
Creating: I-M-MODE-CONTEXT .425
Specializing: I-M-MODE-CONTEXT .425
Recognizing: M-CONNECT
Creating: I-M-CONNECT .420
Specializing: I-M-CONNECT .427
Removing: I-M-IP.389.424
Creating: I-M-IP.389.424
Specializing: I-M-IP.389.424
Recognizing: I-M-TERMINATOR.323
(LOSS * OF OUTPUT - FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

Reading LOSS
Recognizing: M-IP.371
Creating: I-M-IP.371.428
Specializing: I-M-IP.371.428
Recognizing: I-M-TERMINATOR.323
(LOSS * OF OUTPUT - FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

Reading OF
(LOSS OF * OUTPUT - FAILURE TO START) = M-STARTUP-NO-OUTPUT
(LOSS OF * OUTPUT - FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

Reading NAME
(ITEM NAME * :COLON*) = M-ITEM-CONTEXT
(ITEM NAME * :COLON*) = M-ITEM-CONTEXT

Reading OUTPUT
 (LOSS OF OUTPUT * - FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT
 (LOSS OF OUTPUT * - FAILURE TO START) = M-STARTUP-NO-OUTPUT

Reading -
 (LOSS OF OUTPUT * - FAILURE TO START) = M-STARTUP-NO-OUTPUT
 (LOSS OF OUTPUT * - FAILURE DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT

Reading FAILURE
 (LOSS OF OUTPUT - FAILURE * DURING OPERATION) = M-OPERATIONAL-NO-OUTPUT
 (LOSS OF OUTPUT - FAILURE * TO START) = M-STARTUP-NO-OUTPUT

Reading TO
 (LOSS OF OUTPUT - FAILURE TO * START) = M-STARTUP-NO-OUTPUT

Reading START
 (LOSS OF OUTPUT - FAILURE TO START *) = M-STARTUP-NO-OUTPUT referenced

Recognizing: M-STARTUP-NO-OUTPUT
 Creating: I-M-IP.374.439
 Specializing: I-M-IP.374.439
 Recognizing: I-M-TERMINATOR.323
 (CONTAMINATION *) = M-CONTAMINATION-CAUSE referenced

Recognizing: M-CONTAMINATION-CAUSE
 Creating: I-M-IP.374.439
 Specializing: I-M-CONTAMINATION-CAUSE.440
 Specifying: I-M-HL-PATTERN.387
 Recognizing: I-M-HL-PATTERN.387.441
 Specializing: I-M-HL-PATTERN.387.441
 Removing: I-M-IP.374.439

Reading *COMMA*

Reading FAILURES
 (FAILURE CAUSES * COLON *) = M-CAUSE-CONTEXT referenced

Recognizing: M-CAUSE-CONTEXT
 Creating: I-M-IP.390.432
 Specializing: I-M-IP.390.432
 Recognizing: I-M-TERMINATOR.323
 (FAILURE CAUSES * *COLON*) = M-CAUSE-CONTEXT

Reading *COLON*
 (FAILURE CAUSES *COLON*) = M-CAUSE-CONTEXT referenced

Recognizing: M-CAUSE-CONTEXT
 Creating: I-M-CAUSE-CONTEXT.433
 Specializing: I-M-CAUSE-CONTEXT.433
 Recognizing: M-CONTEXT
 Removing: I-M-CONTEXT.427
 Creating: I-M-CONTEXT.435
 Specializing: I-M-CONTEXT.435
 Removing: I-M-IP.390.432

Reading PIECE-PART
 Recognizing: M-IP.373
 Creating: I-M-IP.373.436
 Specializing: I-M-IP.373.436
 Recognizing: I-M-TERMINATOR.323
 (PIECE-PART * FAILURES) = M-FAULTY-COMPONENT-CAUSE

Reading FAILURES
 (PIECE-PART FAILURES *) = M-FAULTY-COMPONENT-CAUSE referenced

Recognizing: M-FAULTY-COMPONENT-CAUSE
 Creating: I-M-FAULTY-COMPONENT-CAUSE.437
 Specializing: I-M-FAULTY-COMPONENT-CAUSE.437
 Recognizing: M-HL-PATTERN.387
 Creating: I-M-HL-PATTERN.387.438
 Specializing: I-M-HL-PATTERN.387.438
 Removing: I-M-IP.373.436

Reading *COMMA*

Reading MECHANICAL
 (MECHANICAL SHOCK *) = M-MECHANICAL-SHOCK-CAUSE referenced

Recognizing: M-IP.376.445
 Creating: I-M-IP.376.445
 Specializing: I-M-IP.376.445
 Recognizing: I-M-TERMINATOR.323
 (MECHANICAL * SHOCK) = M-MECHANICAL-SHOCK-CAUSE

Reading SHOCK
 (MECHANICAL SHOCK *) = M-MECHANICAL-SHOCK-CAUSE referenced

Recognizing: M-IP.376.446
 Creating: I-M-IP.376.446

Reading *COMMA*

Reading CONTAMINATION
 Recognizing: M-IP.374
 Creating: I-M-IP.374.439
 Specializing: I-M-IP.374.439
 Recognizing: I-M-TERMINATOR.323
 (CONTAMINATION *) = M-CONTAMINATION-CAUSE referenced

Recognizing: M-CONTAMINATION-CAUSE
 Creating: I-M-IP.375.442
 Specializing: I-M-IP.375.442
 Specifying: I-M-HL-PATTERN.387
 Recognizing: I-M-HL-PATTERN.387.441
 Specializing: I-M-HL-PATTERN.387.441
 Removing: I-M-IP.374.439

Reading *COMMA*

Reading TEMPERATURE
 Recognizing: M-IP.375
 Creating: I-M-IP.375.442
 Specializing: I-M-IP.375.442
 Specifying: I-M-HL-PATTERN.387
 Recognizing: I-M-HL-PATTERN.387.441
 Specializing: I-M-HL-PATTERN.387.441
 Removing: I-M-IP.374.439

Reading *LEFT-PAREN*
 (TEMPERATURE * LEFT-PAREN* HIGH OR LOW * RIGHT-PAREN*) = M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading HIGH
 (TEMPERATURE * LEFT-PAREN* HIGH * OR LOW * RIGHT-PAREN*) = M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading OR
 (TEMPERATURE * LEFT-PAREN* HIGH OR * LOW * RIGHT-PAREN*) = M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading LOW
 (TEMPERATURE * LEFT-PAREN* HIGH OR LOW * RIGHT-PAREN*) = M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading *RIGHT-PAREN*
 (TEMPERATURE * LEFT-PAREN* HIGH OR LOW * RIGHT-PAREN*) = M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading *RIGHT-PAREN*
 (TEMPERATURE * LEFT-PAREN* HIGH OR LOW * RIGHT-PAREN*) = M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading *RIGHT-PAREN*
 (TEMPERATURE * LEFT-PAREN* HIGH OR LOW * RIGHT-PAREN*) = M-TEMPERATURE-OUT-OF-RANGE-CAUSE

Reading MECHANICAL
 Recognizing: M-IP.376
 Creating: I-M-IP.376.445
 Specializing: I-M-IP.376.445
 Recognizing: I-M-TERMINATOR.323
 (MECHANICAL * SHOCK) = M-MECHANICAL-SHOCK-CAUSE

Reading SHOCK
 (MECHANICAL SHOCK *) = M-MECHANICAL-SHOCK-CAUSE referenced

Recognizing: M-IP.376.446
 Creating: I-M-IP.376.446

Reading *COMMA*

Specializing: I-M-MECHANICAL-SHOCK-CAUSE.446
Recognizing: M-HL-PATTERN.387
Creating: I-M-HL-PATTERN.377.447
Specializing: I-M-HL-PATTERN.387.447
Removing: I-M-IP.376.445

Reading *COLON*
Reading THERMAL
Recognizing: M-IP.377
Creating: I-M-IP.377.448
Specializing: I-M-IP.377.448
Recognizing: I-M-IP.377.448
(THERMAL * SHOCK) = M-THERMAL-SHOCK-CAUSE

Reading SHOCK
(THERMAL SHOCK *) = M-THERMAL-SHOCK-CAUSE referenced
Recognizing: M-THERMAL-SHOCK-CAUSE.449
Creating: I-M-THERMAL-SHOCK-CAUSE.449
Specializing: M-HL-PATTERN.387
Recognizing: I-M-HL-PATTERN.387.450
Creating: I-M-HL-PATTERN.387.450
Specializing: I-M-HL-PATTERN.387.450
Removing: I-M-IP.377.448

Reading FAILURE

Reading DETECTION
Recognizing: M-IP.405
Creating: I-M-IP.405.451
Specializing: I-M-IP.405.451
Recognizing: I-M-TERMINATOR.406
(FAILURE DETECTION * *SLASH* VERIFICATION *COLON*) = M-FAILURE-DETECTION
(SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP) = M-FAILURE-DETECTION
Recognizing: M-IP.391
Creating: I-M-IP.391.452
Specializing: I-M-IP.391.452
Recognizing: I-M-TERMINATOR.323
(FAILURE DETECTION * *SLASH* VERIFICATION *COLON*) = M-DETECTION-CONTEXT
Recognizing: M-IP.380
Creating: I-M-IP.380.453
Specializing: I-M-IP.380.453
Recognizing: I-M-TERMINATOR.323
(DETECTION * PROCEDURE *COLON*) = M-DETECTION-CONTEXT
(SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP) = M-DETECTION-CONTEXT

Reading *SLASH*

(FAILURE DETECTION *SLASH* * VERIFICATION *COLON*) = M-FAILURE-DETECTION
(FAILURE DETECTION *SLASH* * VERIFICATION *COLON*) = M-FAILURE-DETECTION
(SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP) = M-FAILURE-DETECTION

Reading VERIFICATION

(FAILURE DETECTION *SLASH* VERIFICATION *COLON* *) = M-DETECTION-CONTEXT
(FAILURE DETECTION *SLASH* VERIFICATION *COLON*) = M-DETECTION-CONTEXT
(FAILURE DETECTION *SLASH* VERIFICATION *COLON*) = M-DETECTION-CONTEXT
(SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP) = M-DETECTION-CONTEXT

Reading *COLON*

(FAILURE DETECTION *SLASH* VERIFICATION *COLON* *) = M-DETECTION-CONTEXT referenced
Recognizing: M-DETECTION-CONTEXT
Creating: I-M-DETECTION-CONTEXT.454
Specializing: I-M-DETECTION-CONTEXT.454
Recognizing: M-CONTEXT
Removing: I-M-CONTEXT.435

Creating: I-M-CONTEXT.456
Specializing: I-M-CONTEXT.456
Removing: I-M-IP.391.452
(FAILURE DETECTION *SLASH* VERIFICATION *COLON* *) = M-FAILURE-DETECTION
(SEQUENCE-OF-STEPS M-DETECTION-AND-GROUP) = M-FAILURE-DETECTION

Reading INDICATION

Recognizing: M-IP.407
Creating: I-M-IP.407.457
Specializing: I-M-IP.407.457
Recognizing: I-M-TERMINATOR.323
(INDICATION * OF A (FAILED-COMPONENT M-ORU) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) = M-NEXT-NODE-DETECTION

Reading OF

(INDICATION OF * A (FAILED-COMPONENT M-ORU) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) = M-NEXT-NODE-DETECTION

Reading A

Recognizing: M-IP.395
Creating: I-M-IP.395.458
Specializing: I-M-IP.395.458
Recognizing: I-M-TERMINATOR.323
(*LEFT-PAREN* A * *RIGHT-PAREN*) = M-ENUMERATION
(INDICATION OF A * (FAILED-COMPONENT M-ORU) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) = M-NEXT-NODE-DETECTION

Reading RC

Recognizing: M-IP.399
Creating: I-M-IP.399.459
Specializing: I-M-IP.399.459
Recognizing: I-M-TERMINATOR.323
(RC *) = M-RC referenced
Recognizing: M-RC
Creating: I-M-RC.460
Specializing: I-M-RC.460
(INDICATION OF A (FAILED-COMPONENT M-ORU) * FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) = M-NEXT-NODE-DETECTION
Removing: I-M-IP.399.459

Reading FAILURE

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) = M-NEXT-NODE-DETECTION

Reading IS

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) = M-NEXT-NODE-DETECTION

Reading FIRST

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED * BY THE (DETECTION-COMPONENT M-ORU) *PERIOD*) = M-NEXT-NODE-DETECTION

Reading BY

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY * (DETECTION-COMPONENT M-ORU) *PERIOD*) -
M-NEXT-NODE-DETECTION

Reading THE
(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY THE * (DETECTION-COMPONENT M-ORU) * PERIOD*) -
M-NEXT-NODE-DETECTION

Reading *QUOTE*

Reading NEXT
Recognizing: M-IP.408
Creating: I-M-IP.408.466
Specializing: I-M-IP.408.466
Recognizing: I-M-TERMINATOR.323
(*QUOTE* NEXT *QUOTE* ACTIVE NODE ON THE NETWORK) = M-NEXT-RC
Recognizing: M-IP.328
Creating: I-M-IP.328.467
Specializing: I-M-IP.328.467
Recognizing: I-M-TERMINATOR.323
(NEXT * ACTIVE RING CONCENTRATOR IN THE NETWORK) = M-NEXT-RC

Reading *QUOTE*
(*QUOTE* NEXT *QUOTE* * ACTIVE NODE ON THE NETWORK) = M-NEXT-RC

Reading ACTIVE
(*QUOTE* NEXT *QUOTE* ACTIVE * NODE ON THE NETWORK) = M-NEXT-RC

Reading NODE
(*QUOTE* NEXT *QUOTE* ACTIVE NODE * ON THE NETWORK) = M-NEXT-RC

Reading ON
(*QUOTE* NEXT *QUOTE* ACTIVE NODE ON * THE NETWORK) = M-NEXT-RC

Reading THE
(*QUOTE* NEXT *QUOTE* ACTIVE NODE ON THE * NETWORK) = M-NEXT-RC

Reading NETWORK
Recognizing: M-IP.347

Creating: I-M-IP.347.470
Specializing: I-M-IP.347.470

Recognizing: I-M-TERMINATOR.323
(DNS NETWORK *) = M-CORE-NETWORK referenced

Recognizing: M-CORE-NETWORK

Creating: I-M-CORE-NETWORK.471
Specializing: I-M-CORE-NETWORK.471

Removing: I-M-IP.347.470

(*QUOTE* NEXT *QUOTE* ACTIVE NODE ON THE NETWORK *) = M-NEXT-RC referenced

Recognizing: M-NEXT-RC
Creating: I-M-NEXT-RC.472
Specializing: I-M-NEXT-RC.472

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT M-ORU) * PERIOD*) =
M-NEXT-NODE-DETECTION

Removing: I-M-IP.408.466

Reading *PERIOD*

(INDICATION OF A (FAILED-COMPONENT I-M-RC.460) FAILURE IS FIRST
DETECTED BY THE (DETECTION-COMPONENT I-M-TERMINATOR.411
PERIOD) = M-NEXT-NODE-DETECTION referenced

Recognizing: M-NEXT-NODE-DETECTION
Creating: I-M-NEXT-NODE-DETECTION.474
Specializing: I-M-NEXT-NODE-DETECTION.474

Recognizing: M-IP.410
Creating: I-M-IP.410.475
Specializing: I-M-IP.410.475
Recognizing: I-M-TERMINATOR.411
((PROCEDURE M-FAILURE-DETECTION) * (EVENT M-EVENT-STEP)) -
M-DISAMBIGUATE-DETECTION-EVENT

Recognizing: M-HL-PATTERN.409

Creating: I-M-HL-PATTERN.409.476

Specializing: I-M-HL-PATTERN.409.476

Recognizing: M-HL-PATTERN.387

Creating: I-M-HL-PATTERN.387.477

Specializing: I-M-HL-PATTERN.416

Recognizing: M-HL-PATTERN.416.478

Creating: I-M-HL-PATTERN.416.478

Specializing: I-M-HL-PATTERN.416.478

Removing: I-M-IP.407.457

Removing: I-M-IP.328.467

Removing: I-M-IP.395.458

Removing: I-M-IP.380.453

Removing: I-M-IP.371.428

Reading SYSTEM
Recognizing: M-IP.342

Creating: I-M-IP.342.479

Specializing: I-M-IP.342.479

Recognizing: I-M-TERMINATOR.323

(SYSTEM *) = M-SYSTEMS referenced

Recognizing: M-SYSTEMS

Creating: I-M-SYSTEMS.480

Specializing: I-M-SYSTEMS.480

Recognizing: M-IP.415

Creating: I-M-IP.415.481

Specializing: I-M-IP.415.481

Recognizing: I-M-IP.342.479

Creating: I-M-IP.342.479

Specializing: I-M-TERMINATOR.323

((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*,

(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

(SYSTEM * MANAGEMENT) = M-SM

Removing: I-M-IP.342.479

Recognizing: M-IP.338

Creating: I-M-IP.338.482

Specializing: I-M-IP.338.482

Recognizing: I-M-TERMINATOR.323

(SYSTEM MANAGEMENT) = M-SM

Removing: I-M-IP.338.482

Reading WILL

Reading REACH

Reading A

Recognizing: M-IP.395

Creating: I-M-IP.395.484

Specializing: I-M-IP.395.484

(*LEFT-PAREN* A * RIGHT-PAREN*) = M ENUMERATION

Reading TIME-OUT

Recognizing: M-IP.358

Creating: I-M-IP.358.485

Specializing: I-M-IP.358.485

```

Specializing: I-M-DISAMBIGUATE-DETECTION-EVENT.506
Creating: I-M-TIME-OUT-EVENT.144.507
Recognizing: I-M-TIME-OUT-EVENT.144.507
Recognizing: M-HL-PATTERN.416
Creating: I-M-HL-PATTERN.416.527
Specializing: I-M-HL-PATTERN.416.527
Creating: I-M-HL-PATTERN.416.478
Removing: I-M-HL-PATTERN.416.478
Recognizing: M-HL-PATTERN.416
Creating: I-M-HL-PATTERN.416.528
Specializing: I-M-HL-PATTERN.416.528
Creating: I-M-HL-PATTERN.416.505
Removing: I-M-HL-PATTERN.416.505
Recognizing: M-HL-PATTERN.416
Creating: I-M-HL-PATTERN.416.529
Removing: I-M-HL-PATTERN.416.529
Specializing: I-M-HL-PATTERN.416.529
Creating: I-M-HL-PATTERN.416.528
Specializing: I-M-HL-PATTERN.416.528
Creating: I-M-HL-PATTERN.416.475
Removing: I-M-HL-PATTERN.416.475
Recognizing: M-HL-PATTERN.416
Creating: I-M-HL-PATTERN.416.481
Removing: I-M-HL-PATTERN.416.481
Specializing: I-M-HL-PATTERN.416.481
Creating: I-M-HL-PATTERN.416.485
Removing: I-M-HL-PATTERN.416.485
Specializing: I-M-HL-PATTERN.416.485
Creating: I-M-HL-PATTERN.416.481
Removing: I-M-HL-PATTERN.416.481

Reading CORRECTIVE
  Recognizing: M-IP.392
  Creating: I-M-IP.392.530
  Specializing: I-M-IP.392.530
  Recognizing: I-M-TERMINATOR.323
  (CORRECTIVE * ACTION * COLON*) = M-CORRECTION-CONTEXT

Reading ACTION
  (CORRECTIVE ACTION * *COLON*) = M-CORRECTION-CONTEXT
  Reading *COLON*
    (CORRECTIVE ACTION *COLON* *) = M-CORRECTION-CONTEXT
    Recognizing: M-CORRECTION-CONTEXT
    Creating: I-M-CORRECTION-CONTEXT-CONTINUATION.531
    Specializing: I-M-CORRECTION-CONTEXT.531
    Recognizing: M-CONTEXT
    Creating: I-M-CONTEXT.456
    Removing: I-M-CONTEXT.456
    Specializing: I-M-AND-GROUP.145.534
    Recognizing: M-BUILD-PROCEDURE
    Creating: I-M-BUILD-PROCEDURE.533
    Specializing: I-M-BUILD-PROCEDURE.533
    Recognizing: M-AND-GROUP.145
    Creating: I-M-AND-GROUP.145.534
    Removing: I-M-HL-PATTERN.416.505
    Specializing: I-M-HL-PATTERN.416.505
    Recognizing: M-DETECTION-PROC.146.558
    Creating: I-M-DETECTION-PROC.146.558
    Specializing: I-M-DETECTION-PROC.146.558
    Recognizing: M-BUILD-PROCEDURE
    Creating: I-M-BUILD-PROCEDURE.560
    Specializing: M-BUILD-PROCEDURE.560
    Recognizing: M-DETECTION-AND-GROUP.147
    Creating: I-M-DETECTION-AND-GROUP.147.561
    Specializing: I-M-DETECTION-AND-GROUP.147.561
    Recognizing: M-BUILD-PROCEDURE
    Creating: I-M-BUILD-PROCEDURE.588
    Specializing: I-M-BUILD-PROCEDURE.588
    Recognizing: M-NEXT-NODE-DETECTION
    Creating: I-M-NEXT-NODE-DETECTION.589
    Specializing: I-M-NEXT-NODE-DETECTION.589
    Recognizing: M-IP.410.590
    Creating: I-M-IP.410.590
Specializing: M-DISAMBIGUATE-DETECTION-EVENT
  Referenced
    Recognizing: M-TIME-OUT-EVENT.492
    Creating: I-M-TIME-OUT-EVENT.492
    Specializing: I-M-TIME-OUT-EVENT.492
    (PROCEDURE I-M-DISAMBIGUATE-DETECTION-EVENT
      (EVENT I-M-TIME-OUT-EVENT.492 *) -
        M-DISAMBIGUATE-DETECTION-EVENT referenced
    Creating: I-M-DISAMBIGUATE-DETECTION-EVENT
    Specializing: I-M-DISAMBIGUATE-DETECTION-EVENT
    Creating: I-M-DISAMBIGUATE-DETECTION-EVENT.506

```

Recognizing: I-M-TERMINATOR.411 * (EVENT M-EVENT-STEP) -
 ((PROCEDURE M-FAILURE-DETECTION) * (EVENT M-EVENT-STEP)) -
 M-DISAMBIGUATE-DETECTION-EVENT
 Recognizing: M-HL-PATTERN.409
 Creating: I-M-HL-PATTERN.409.591
 Specializing: I-M-HL-PATTERN.409.591
 Recognizing: M-HL-PATTERN.387
 Creating: I-M-HL-PATTERN.387.592
 Specializing: I-M-HL-PATTERN.387.592
 Recognizing: M-BUILD-PROCEDURE
 Creating: I-M-BUILD-PROCEDURE.594
 Specializing: I-M-BUILD-PROCEDURE.594
 Recognizing: I-M-HL-PATTERN.409.591
 Removing: I-M-HL-PATTERN.409.476
 Recognizing: M-UNIFY-DETECTION-MOPS
 Creating: I-M-UNITY-DETECTION-MOPS.595
 Specializing: I-M-UNITY-DETECTION-MOPS.595
 Creating: I-M-NEXT-NODE-DETECTION.596
 Recognizing: I-M-NEXT-NODE-DETECTION.596
 Specializing: M-IP.410
 Recognizing: I-M-IP.410.597
 Specializing: I-M-IP.410.597
 Recognizing: I-M-TERMINATOR.411
 ((PROCEDURE M-FAILURE-DETECTION) * (EVENT M-EVENT-STEP)) -
 M-DISAMBIGUATE-DETECTION-EVENT
 Creating: M-UNIFY-DETECTION-MOPS
 Specializing: I-M-UNIFY-DETECTION-MOPS.599
 Specializing: I-M-UNIFY-DETECTION-MOPS.599
 Recognizing: M-HL-PATTERN.387
 Creating: I-M-HL-PATTERN.387.600
 Specializing: I-M-HL-PATTERN.387.600
 Recognizing: M-BUILD-PROCEDURE
 Creating: I-M-BUILD-PROCEDURE.602
 Specializing: I-M-BUILD-PROCEDURE.602
 Creating: I-M-CONTEXT.603
 Specializing: I-M-CONTEXT.603
 Removing: I-M-IP.392.530
 Removing: I-M-IP.410.597
 Removing: I-M-IP.410.590

Reading *LEFT-PAREN*
 Recognizing: M-IP.395
 Creating: I-M-IP.395.604
 Specializing: I-M-ENUMERATION.604
 Recognizing: I-M-TERMINATOR.323
 (*LEFT-PAREN* A *RIGHT-PAREN*) - M-ENUMERATION

Reading *RIGHT-PAREN*
 Recognizing: I-M-IP.395.604
 Specializing: M-ENUMERATION.605
 Recognizing: M-BUILD-CONTEXT
 Creating: I-M-BUILD-CONTEXT.606
 Specializing: I-M-BUILD-CONTEXT.606
 Recognizing: M-CONTEXT
 Removing: I-M-CONTEXT.603
 Creating: I-M-CONTEXT.608
 Specializing: I-M-CONTEXT.608
 Removing: I-M-IP.395.604

Reading SHORT

Recognizing: M-IP.382
 Creating: I-M-IP.382.609
 Specializing: I-M-IP.382.609
 Recognizing: I-M-TERMINATOR.323
 (SHORT * TERM) - M-SHORT-TERM

Reading TERM
 (SHORT TERM *) - M-SHORT-TERM referenced
 Recognizing: M-SHORT-TERM
 Creating: I-M-SHORT-TERM.610
 Specializing: I-M-SHORT-TERM.610
 NIL
 Recognizing: M-IP.394
 Creating: I-M-IP.394.611
 Specializing: I-M-IP.394.611
 Recognizing: I-M-TERMINATOR.323
 ((FRAMEWORK M-FRAMEWORK) * *COLON*) - M-FAILURE-CORRECTION
 Removing: I-M-IP.382.609

Reading *COLON*
 ((FRAMEWORK I-M-SHORT-TERM.610) *COLON*) - M-FAILURE-CORRECTION referenced
 Recognizing: M-FAILURE-CORRECTION
 Creating: I-M-FAILURE-CORRECTION.612
 Specializing: I-M-FAILURE-CORRECTION.612
 Recognizing: M-HL-PATTERN.404
 Creating: I-M-HL-PATTERN.404.613
 Specializing: I-M-HL-PATTERN.404.613
 Recognizing: M-HL-PATTERN.387
 Creating: I-M-HL-PATTERN.387.614
 Specializing: I-M-HL-PATTERN.387.614
 Recognizing: M-HL-PATTERN.416
 Creating: I-M-HL-PATTERN.416.615
 Specializing: I-M-HL-PATTERN.416.615
 Removing: I-M-IP.394.611

Reading NETWORK
 Recognizing: M-IP.347
 Creating: I-M-IP.347.645
 Specializing: I-M-HL-PATTERN.387.614
 Recognizing: I-M-TERMINATOR.323
 (DMS NETWORK *) - M-CORE-NETWORK referenced
 Recognizing: M-CORE-NETWORK
 Removing: I-M-CORE-NETWORK.646
 Specializing: I-M-CORE-NETWORK.646
 Removing: I-M-IP.347.645

Reading RECONFIGURATION
 Recognizing: M-IP.401
 Creating: I-M-IP.401.647
 Specializing: I-M-IP.401.647
 Recognizing: I-M-TERMINATOR.323
 ((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION * IS EFFECTED
 AUTOMATICALLY *PERIOD*) - M-AUTO-RECONFIG

Reading IS
 ((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION IS * EFFECTED
 AUTOMATICALLY *PERIOD*) - M-AUTO-RECONFIG

Reading EFFECTED
 ((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION IS EFFECTED
 AUTOMATICALLY *PERIOD*) - M-AUTO-RECONFIG

Reading AUTOMATICALLY
 ((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION IS EFFECTED
 AUTOMATICALLY *PERIOD*) - M-AUTO-RECONFIG

Reading *PERIOD*
 ((CONFIGURATION I-M-CORE-NETWORK.471) RECONFIGURATION IS EFFECTED
 AUTOMATICALLY *PERIOD* *) = M-AUTO-RECONFIG referenced

Recognizing: M-AUTO-RECONFIG
 Creating: I-M-IP.402.648
 Specializing: I-M-AUTO-RECONFIG.648

Inferred--> M-RECONFIGURATION
 Recognizing: M-RECONFIGURATION.649
 Creating: I-M-RECONFIGURATION.649
 Specializing: M-HL-PATTERN.404

Recognizing: I-M-HL-PATTERN.404.650
 Creating: I-M-HL-PATTERN.404.650
 Specializing: M-HL-PATTERN.387

Recognizing: M-HL-PATTERN.387.651
 Creating: I-M-HL-PATTERN.387.651
 Specializing: M-HL-PATTERN.416

Recognizing: M-HL-PATTERN.416.652
 Creating: I-M-HL-PATTERN.416.652
 Removing: I-M-IP.401.647

Reading THE
 Recognizing: M-IP.471
 Creating: I-M-IP.471
 Specializing: I-M-IP.471

Reading DMS
 Recognizing: M-IP.471
 Creating: I-M-IP.471
 Specializing: I-M-IP.471

Reading NETWORK
 Recognizing: M-IP.347
 Creating: I-M-IP.347.667
 Specializing: I-M-IP.347.667

Recognizing: I-M-TERMINATOR.323
 (DMS NETWORK *) = M-CORE-NETWORK referenced

Recognizing: M-CORE-NETWORK
 Removing: I-M-CORE-NETWORK.668
 Specializing: I-M-CORE-NETWORK.471
 Removing: I-M-IP.347.667

Reading REMAINS
 Recognizing: M-IP.403
 Creating: I-M-IP.403.669
 Specializing: I-M-IP.403.669

Recognizing: I-M-TERMINATOR.323
 (THE (OBJECT I-M-CORE-NETWORK.471) REMAINS * IN OPERATION) = M-OPERATIONAL

Reading IN
 (THE (OBJECT I-M-CORE-NETWORK.471) REMAINS IN * OPERATION *) =
 M-OPERATIONAL referenced

Recognizing: M-OPERATIONAL
 Creating: I-M-OPERATIONAL.670
 Specializing: I-M-OPERATIONAL.670

Removing: I-M-CORE-NETWORK.671
 Removing: I-M-IP.403.669

Reading OPERATION
 (THE (OBJECT I-M-CORE-NETWORK.471) REMAINS IN OPERATION *) =
 M-OPERATIONAL

Reading IN
 (THE (OBJECT I-M-CORE-NETWORK.471) REMAINS IN OPERATION *) =
 M-OPERATIONAL referenced

Recognizing: M-OPERATIONAL
 Creating: I-M-OPERATIONAL.670
 Specializing: I-M-OPERATIONAL.670

Removing: I-M-CORE-NETWORK.671
 Removing: I-M-IP.403.669

Reading IN
 Reading A
 Recognizing: M-IP.395
 Creating: I-M-IP.395.672
 Specializing: I-M-IP.395.672

Recognizing: I-M-TERMINATOR.323
 (*LEFT-PAREN* A * *RIGHT-PAREN*) = M-ENUMERATION

Recognizing: I-M-TERMINATOR.323
(*LEFT-PAREN* B *RIGHT-PAREN*) = M-ENUMERATION

Reading *RIGHT-PAREN*

Recognizing: M-ENUMERATION

Removing: I-M-ENUMERATION .685

Specializing: I-M-ENUMERATION .605

Recognizing: M-BUILD-CONTEXT

Removing: I-M-BUILD-CONTEXT.687

Specializing: I-M-BUILD-CONTEXT .606

Recognizing: M-CONTEXT

Removing: I-M-CONTEXT .608

Recognizing: I-M-HL-PATTERN .416.684

Removing: I-M-HL-PATTERN .416.652

Removing: I-M-HL-PATTERN .416.615

Recognizing: M-BUILD-PROCEDURE

Creating: I-M-BUILD-PROCEDURE .689

Specializing: I-M-BUILD-PROCEDURE .689

Recognizing: I-M-HL-PATTERN .401.682

Removing: I-M-HL-PATTERN .404.650

Removing: I-M-HL-PATTERN .404.613

Recognizing: M-UNITY-CORRECTION-MOPS .690

Creating: I-M-UNITY-CORRECTION-MOPS .690

Specializing: I-M-UNITY-CORRECTION-MOPS .690

Creating: I-M-BYPASS-NODE .691

Recognizing: M-UNITY-CORRECTION-MOPS .691

Creating: I-M-UNITY-CORRECTION-MOPS .693

Specializing: I-M-UNITY-CORRECTION-MOPS .693

Creating: I-M-HL-PATTERN .387

Creating: I-M-HL-PATTERN .387.694

Specializing: I-M-HL-PATTERN .387.694

Recognizing: M-BUILD-PROCEDURE

Creating: I-M-BUILD-PROCEDURE .696

Specializing: I-M-BUILD-PROCEDURE .696

Creating: I-M-CONTEXT .697

Specializing: I-M-CONTEXT .697

Removing: I-M-IP.396.685

Reading LONG

Recognizing: M-IP.383

Creating: I-M-IP.383.698

Specializing: I-M-LONG-TERM .698

Recognizing: I-M-TERMINATOR.323

(LONG * TERM) = M-LONG-TERM

Reading TERM

(LONG TERM *) = M-LONG-TERM referenced

Recognizing: M-LONG-TERM

Creating: I-M-LONG-TERM .699

Specializing: I-M-LONG-TERM .699

Recognizing: M-IP.394

Creating: I-M-IP.394.700

Specializing: I-M-IP.394.700

Recognizing: I-M-TERMINATOR.323

((FRAMEWORK M-FRAMEWORK) * *COLON*) = M-FAILURE-CORRECTION

Removing: I-M-IP.383.698

Reading *COLON*

((FRAMEWORK I-M-LONG-TERM.699) *COLON*) = M-FAILURE-CORRECTION referenced

Recognizing: M-FAILURE-CORRECTION

Creating: I-M-FAILURE-CORRECTION.701

Specializing: I-M-FAILURE-CORRECTION .701

Recognizing: M-HL-PATTERN .404

Creating: I-M-HL-PATTERN .407.702

Specializing: I-M-HL-PATTERN .404.702

Recognizing: M-HL-PATTERN .387

Creating: I-M-HL-PATTERN .387.703

Specializing: I-M-HL-PATTERN .387.703

Recognizing: M-HL-PATTERN .416

Creating: I-M-HL-PATTERN .416.704

Specializing: I-M-HL-PATTERN .416.704

Removing: I-M-IP.394.700

Removing: I-M-IP.394.700

Reading THE

Reading CREWMEN

Recognizing: M-IP.344

Creating: I-M-IP.344.736

Specializing: I-M-IP.344.736

Recognizing: I-M-TERMINATOR.323

(THE CREWMEN *) = M-CREW referenced

Recognizing: M-CREW

Creating: I-M-CREW .737

Specializing: I-M-CREW .737

Recognizing: I-M-CREW .737

Creating: I-M-IP.415

Specializing: I-M-IP.415.738

Creating: I-M-IP.415.738

Specializing: I-M-IP.415.738

Recognizing: I-M-TERMINATOR.323

((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*

(SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Removing: I-M-IP.344.736

Reading CHECK

Recognizing: M-IP.368

Creating: I-M-IP.368.739

Specializing: I-M-IP.368.739

Recognizing: I-M-IP.345.740

Creating: I-M-IP.345.740

Specializing: I-M-IP.345.740

Recognizing: I-M-CREW .737

((ACTOR I-M-CREW .737) CHECK * FOR (OBJECT M-SYSTEM)) = M-VERIFY-EVENT

Reading FOR

((ACTOR I-M-CREW .737) CHECK FOR * (OBJECT M-SYSTEM)) = M-VERIFY-EVENT

Reading *QUOTE*

Reading APPLIED

Recognizing: M-IP.345

Creating: I-M-IP.345.741

Specializing: I-M-IP.345.741

Recognizing: I-M-TERMINATOR.323

(*QUOTE* APPLIED * POWER *QUOTE*) = M-POWER-SYSTEM

(ACTOR I-M-CREW .737) CHECK FOR (OBJECT I-M-POWER-SYSTEM .308) * -

M-VERIFY-EVENT referenced

Recognizing: M-VERIFY-EVENT

Creating: I-M-VERIFY-EVENT .742

Specializing: I-M-VERIFY-EVENT .742

Recognizing: M-HL-PATTERN .416

Creating: I-M-HL-PATTERN .416.745

Creating: I-M-HL-PATTERN .416.745

Specializing: I-M-HL-PATTERN.416.745
Removing: I-M-IP.368.739
Removing: I-M-IP.345.740

Reading AND

Reading THE

Reading CREWMEN

Recognizing: M-IP.344
Creating: I-M-IP.344.748
Specializing: I-M-IP.344.748
Recognizing: I-M-TERMINATOR.323
(THE CREWMEN *) = M-CREW referenced

Recognizing: M-CREW

Removing: I-M-CREW.749
Specializing: I-M-CREW.737
Recognizing: M-IP.415
Creating: I-M-IP.415.750
Specializing: I-M-IP.415.750
Recognizing: I-M-TERMINATOR.323
(EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLOR*
(SEQUENCE-OF-STEPS M-PROCEDURE) * M-FAILURE-EFFECT
Removing: I-M-IP.344.748

Reading CHECK

Recognizing: M-IP.368
Creating: I-M-IP.368.751
Specializing: I-M-IP.368.751
Recognizing: I-M-TERMINATOR.323
(ACTOR I-M-CREW.737) CHECK * FOR (OBJECT M-SYSTEM) = M-VERIFY-EVENT

Reading FOR

(ACTOR I-M-CREW.737) CHECK FOR * (OBJECT M-SYSTEM) = M-VERIFY-EVENT

Reading *QUOTE*

Reading CONNECTOR
Recognizing: M-IP.346
Creating: I-M-IP.346.752
Specializing: I-M-IP.346.752
Recognizing: I-M-TERMINATOR.323
(*QUOTE* CONNECTOR * TIGHTNESS * QUOTE*) = M-CONNECTORS

Reading TIGHTNESS

(*QUOTE* CONNECTOR TIGHTNESS * QUOTE*) = M-CONNECTORS
Referencing: M-CONNECTORS
Removing: I-M-CONNECTORS.310
(ACTOR I-M-CREW.737) CHECK FOR (OBJECT I-M-CONNECTORS.310) * = M-VERIFY-EVENT referenced

Reading *QUOTE*

Recognizing: M-CONNECTORS
Creating: I-M-CONNECTORS.310
Specializing: I-M-CONNECTORS.310
(ACTOR I-M-CREW.737) CHECK FOR (OBJECT I-M-CONNECTORS.310) * = M-VERIFY-EVENT
Creating: I-M-VERIFY-EVENT.754
Specializing: I-M-VERIFY-EVENT.754
Removing: I-M-CREW.755

Removing: I-M-CONNECTORS.756

Recognizing: M-HL-PATTERN.416
Creating: I-M-HL-PATTERN.416.757
Specializing: I-M-HL-PATTERN.416.757
Removing: I-M-IP.368.751
Removing: I-M-IP.346.752

Reading *PERIOD*

Removing: I-M-IP.415.750
Removing: I-M-IP.415.738

Reading IF

Recognizing: F-UNKNOWN-WORD
Creating: I-F-UNKNOWN-WORD.758
Specializing: I-F-UNKNOWN-WORD.758
Failure: IF is an unknown-word.
Repair: define-unknown-word on IF.
Removing: I-F-UNKNOWN-WORD.758

Reading THE

Reading RC

Recognizing: M-IP.399
Creating: I-M-IP.399.761
Specializing: I-M-IP.399.761
Recognizing: I-M-TERMINATOR.323
(RC *) = M-RC referenced
Recognizing: M-RC
Creating: I-M-RC.762
Specializing: I-M-RC.460
Removing: I-M-IP.399.761

Reading CANNOT

Recognizing: F-UNKNOWN-WORD
Creating: I-F-UNKNOWN-WORD.766
Specializing: I-F-UNKNOWN-WORD.766
Failure: CANNOT is an unknown-word.
Repair: define-unknown-word on CANNOT.
Removing: I-F-UNKNOWN-WORD.766

Reading THEN

Recognizing: F-UNKNOWN-WORD
Creating: I-F-UNKNOWN-WORD.767
Specializing: I-F-UNKNOWN-WORD.767
Failure: THEN is an unknown-word.
Repair: define-unknown-word on THEN.
Removing: I-F-UNKNOWN-WORD.767

Reading BE

Recognizing: F-UNKNOWN-WORD
Creating: I-F-UNKNOWN-WORD.768
Specializing: I-F-UNKNOWN-WORD.768
Failure: BE is an unknown-word.
Repair: define-unknown-word on BE.
Removing: I-F-UNKNOWN-WORD.768

Reading PLACED

Recognizing: F-UNKNOWN-WORD
Creating: I-F-UNKNOWN-WORD.769
Specializing: I-F-UNKNOWN-WORD.769
Failure: PLACED is an unknown-word.
Repair: define-unknown-word on PLACED.
Removing: I-F-UNKNOWN-WORD.769

Reading IN

Reading OPERATION

Reading *COMMA*

Reading THE

Reading RC
Recognizing: M-IP.399
Creating: I-M-IP.399.772
Specializing: I-M-IP.399.772
Recognizing: I-M-TERMINATOR.323
(RC *) = M-RC referenced
Recognizing: M-RC
Removing: I-M-RC.773
Specializing: I-M-RC.460
Removing: I-M-IP.399.772

Reading IS

Reading REMOVED

Reading AND
Reading REPLACED
Recognizing: M-IP.400
Creating: I-M-IP.400.777
Specializing: I-M-IP.400.777
Recognizing: I-M-TERMINATOR.323
(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED * WITH
AN ORO LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading WITH
(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH *
AN ORO LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading AN
(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
* ORU LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading ORU
(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
ORU * LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading LOGISTICS
(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
ORU LOGISTICS * SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Reading SPARE
(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
ORU LOGISTICS SPARE * *PERIOD*) = M-REPLACE-WITH-SPARE referenced

Reading *PERIOD*
(THE (FAILED-COMPONENT I-M-RC.460) IS REMOVED AND REPLACED WITH AN
ORU LOGISTICS SPARE *PERIOD*) = M-REPLACE-WITH-SPARE

Recognizing: I-M-REPLACE-WITH-SPARE
Creating: I-M-REPLACE-WITH-SPARE.778
Specializing: I-M-REPLACE-WITH-SPARE.778
Recognizing: M-HL-PATTERN.404
Creating: I-M-HL-PATTERN.404.779
Specializing: I-M-HL-PATTERN.404.779
Recognizing: M-HL-PATTERN.387
Creating: I-M-HL-PATTERN.387.780
Specializing: I-M-HL-PATTERN.387.780
Recognizing: M-HL-PATTERN.416
Creating: I-M-HL-PATTERN.416.781
Specializing: I-M-HL-PATTERN.416.781
Removing: I-M-IP.400.777

Reading FAILURE

Reading EFFECT
Recognizing: M-IP.393
Creating: I-M-IP.393.782
Specializing: I-M-IP.393.782
Recognizing: I-M-TERMINATOR.323
(FAILURE EFFECT * ON *COLON*) = M-EFFECT-CONTEXT
Recognizing: M-IP.386
Creating: I-M-IP.386.783
Specializing: I-M-IP.386.783
Recognizing: I-M-TERMINATOR.323
(FAILURE EFFECT * ON THE (EFFECTED-COMPONENT M-MISSION-COMPONENT)
COLON (SEQUENCE-OF-STEPS M-GROUP)) = M-FAILURE-EFFECT

Reading ON
(FAILURE EFFECT ON * THE (EFFECTED-COMPONENT M-MISSION-COMPONENT)
COLON (SEQUENCE-OF-STEPS M-GROUP)) = M-EFFECT-CONTEXT
(FAILURE EFFECT ON * *COLON*) = M-EFFECT-CONTEXT

Reading *COLON*
(FAILURE EFFECT ON *COLON*) = M-EFFECT-CONTEXT referenced
Recognizing: M-EFFECT-CONTEXT
Creating: I-M-EFFECT-CONTEXT.784
Specializing: I-M-EFFECT-CONTEXT.784
Recognizing: M-CONTEXT
Removing: I-M-CONTEXT.697
Removing: I-M-HL-PATTERN.416.781
Removing: I-M-HL-PATTERN.416.757
Removing: I-M-HL-PATTERN.416.745
Removing: I-M-HL-PATTERN.416.704
Recognizing: M-BUILD-PROCEDURE
Creating: I-M-BUILD-PROCEDURE.786
Specializing: I-M-BUILD-PROCEDURE.786
Recognizing: M-AND-GROUP.313
Creating: I-M-AND-GROUP.313.787
Specializing: I-M-AND-GROUP.313.787
Recognizing: M-AND-GROUP.314
Creating: I-M-AND-GROUP.314.827
Specializing: I-M-AND-GROUP.314.827
Recognizing: M-BUILD-PROCEDURE
Creating: I-M-BUILD-PROCEDURE.826
Specializing: I-M-BUILD-PROCEDURE.826
Recognizing: M-AND-GROUP.314
Creating: I-M-AND-GROUP.313.787
Specializing: I-M-AND-GROUP.313.787
Recognizing: M-BUILD-PROCEDURE
Creating: I-M-BUILD-PROCEDURE.826
Specializing: I-M-BUILD-PROCEDURE.826
Recognizing: M-AND-GROUP.315
Creating: I-M-CORRECTION-PROC.315.886
Specializing: I-M-CORRECTION-PROC.315.886
Recognizing: M-BUILD-PROCEDURE
Creating: I-M-BUILD-PROCEDURE.886
Specializing: I-M-BUILD-PROCEDURE.886
Recognizing: M-CORRECTION-AND-GROUP.316
Creating: I-M-CORRECTION-AND-GROUP.316.889
Specializing: I-M-CORRECTION-AND-GROUP.316.889
Recognizing: M-BUILD-PROCEDURE
Creating: I-M-BUILD-PROCEDURE.931
Specializing: I-M-BUILD-PROCEDURE.931
Recognizing: M-NON-OPERATIONAL-REPLACEMENT
Creating: I-M-NON-OPERATIONAL-REPLACEMENT.932
Specializing: I-M-NON-OPERATIONAL-REPLACEMENT.932
Recognizing: M-HL-PATTERN.404
Creating: I-M-HL-PATTERN.404.933
Specializing: I-M-HL-PATTERN.404.933

Recognizing: M-HL-PATTERN .387
 Creating: I-M-HL-PATTERN .387 .934
 Specializing: I-M-HL-PATTERN .387 .934
 Recognizing: M-BUILD-PROCEDURE
 Creating: I-M-BUILD-PROCEDURE .936
 Specializing: I-M-BUILD-PROCEDURE .936
 Removing: I-M-HL-PATTERN .404 .933
 Removing: I-M-HL-PATTERN .404 .779
 Removing: I-M-HL-PATTERN .404 .702
 Recognizing: M-UNIFY-CORRECTION-MOPS .937
 Creating: I-M-UNIFY-CORRECTION-MOPS .937
 Specializing: I-M-UNIFY-CORRECTION-MOPS .938
 Creating: I-M-FAILURE-CORRECTION .938
 Recognizing: I-M-FAILURE-CORRECTION .938
 Recognizing: M-HL-PATTERN .387 .939
 Creating: I-M-HL-PATTERN .387 .939
 Specializing: M-UNIFY-CORRECTION-MOPS .941
 Creating: I-M-UNIFY-CORRECTION-MOPS .941
 Specializing: I-M-HL-PATTERN .387 .939
 Creating: I-M-HL-PATTERN .387 .942
 Specializing: I-M-HL-PATTERN .387 .942
 Recognizing: M-BUILD-PROCEDURE
 Creating: I-M-BUILD-PROCEDURE .944
 Specializing: I-M-BUILD-PROCEDURE .944
 Recognizing: M-UNIFY-CORRECTION-MOPS .946
 Removing: I-M-UNIFY-CORRECTION-MOPS .946
 Specializing: I-M-UNIFY-CORRECTION-MOPS .946
 Creating: I-M-HL-PATTERN .387 .947
 Removing: I-M-HL-PATTERN .387 .947
 Specializing: I-M-HL-PATTERN .387 .947
 Recognizing: M-BUILD-PROCEDURE
 Creating: I-M-BUILD-PROCEDURE .949
 Specializing: I-M-BUILD-PROCEDURE .949
 Creating: I-M-HL-PATTERN .387 .950
 Removing: I-M-HL-PATTERN .387 .950
 Specializing: I-M-HL-PATTERN .387 .950
 Recognizing: M-UNIFY-CORRECTION-MOPS .952
 Specializing: I-M-UNIFY-CORRECTION-MOPS .952
 Creating: I-M-HL-PATTERN .387 .953
 Removing: I-M-HL-PATTERN .387 .953
 Specializing: I-M-HL-PATTERN .387 .953
 Creating: I-M-BUILD-PROCEDURE .955
 Specializing: I-M-BUILD-PROCEDURE .955
 Creating: I-M-CONTEXT .956
 Specializing: I-M-CONTEXT .956
 Removing: I-M-IP .393 .782

Reading *LEFT-PAREN*
 Reading A
 Recognizing: M-IP .395
 Creating: I-M-IP .395 .957
 Specializing: I-M-IP .395 .957
 Recognizing: I-M-TERMINATOR .323
 (*LEFT-PAREN* A *RIGHT-PAREN*) = M-ENUMERATION
 Reading *RIGHT-PAREN*
 (*LEFT-PAREN* A *RIGHT-PAREN* *) = M-ENUMERATION referenced
 Recognizing: M-ENUMERATION
 Removing: I-M-ENUMERATION .958

Recognizing: I-M-ENUMERATION .605
 Recognizing: M-BUILD-CONTEXT
 Creating: I-M-BUILD-CONTEXT .959
 Specializing: I-M-BUILD-CONTEXT .959
 Recognizing: I-M-CONTEXT .956
 Removing: I-M-CONTEXT .956
 Creating: I-M-CONTEXT .961
 Specializing: I-M-CONTEXT .961
 Removing: I-M-IP .395 .957

Reading CREW
 Recognizing: M-CREW
 Creating: I-M-IP .318 .962
 Specializing: I-M-CREW .962
 Recognizing: I-M-TERMINATOR .323
 (CREW *) = M-CREW referenced
 Recognizing: M-CREW
 Removing: I-M-CREW .963
 Specializing: I-M-CREW .737
 Recognizing: M-IP .415
 Creating: I-M-IP .415 .964
 Specializing: I-M-IP .415 .964
 Recognizing: I-M-TERMINATOR .323
 ((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
 (SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT
 Removing: I-M-IP .348 .962

Reading *SLASH*
 Reading SSPE
 Recognizing: M-IP .340
 Creating: I-M-IP .340 .965
 Specializing: I-M-IP .340 .965
 Recognizing: I-M-TERMINATOR .323
 (CREW *SLASH* SSPE *) = M-CREW referenced
 Recognizing: M-CREW
 Removing: I-M-CREW .966
 Specializing: I-M-CREW .737
 Recognizing: M-IP .415
 Creating: I-M-IP .415 .967
 Specializing: I-M-IP .415 .967
 Recognizing: I-M-TERMINATOR .323
 ((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
 (SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT
 Removing: I-M-IP .340 .965

Reading *COLON*
 Reading PERIOD
 ((EFFECTED-COMPONENT I-M-CREW .737) *COLON* *
 (SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT
 ((EFFECTED-COMPONENT I-M-CREW .737) *COLON* *
 (SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT
 Reading NONE
 Recognizing: M-IP .414
 Creating: I-M-IP .414 .968
 Specializing: I-M-IP .414 .968
 Recognizing: I-M-TERMINATOR .323
 (NONE * *PERIOD*) = M-DISAMBIGUATE-NONE

Reading *PERIOD*
 ((NONE *PERIOD* *) = M-DISAMBIGUATE-NONE referenced
 Recognizing: M-DISAMBIGUATE-NONE
 Creating: I-M-DISAMBIGUATE-NONE .969
 Specializing: I-M-DISAMBIGUATE-NONE .969

Recognizing: I-M-NULL-PROCEDURE
 ((EFFECTED-COMPONENT I-M-CREW.737) *COLON*
 (SEQUENCE-OF-STEPS I-M-NULL-PROCEDURE) *) = M-FAILURE-EFFECT
 referenced
 Recognizing: M-FAILURE-EFFECT
 Creating: I-M-FAILURE-EFFECT
 Specializing: I-M-FAILURE-EFFECT.970
 Recognizing: M-IP.412
 Creating: I-M-IP.412.971
 Specializing: I-M-IP.412.971
 Recognizing: I-M-TERMINATOR.411
 (SEQUENCE-OF-STEPS M-FAILURE-EFFECT) *
 (JUSTIFICATION M-FAILURE-CORRECTION) * = M-SEARCH-FOR-EFFECT-REFERENT
 Recognizing: M-HL-PATTERN.387
 Creating: I-M-HL-PATTERN.387.972
 Specializing: M-HL-PATTERN.387.972
 Recognizing: M-HL-PATTERN.416
 Creating: I-M-HL-PATTERN.416.973
 Specializing: I-M-HL-PATTERN.416.973
 Removing: I-M-IP.415.964
 Removing: I-M-IP.414.968
 Removing: I-M-IP.415.967
 Removing: I-M-IP.386.783
 Reading THE
 Reading CONDITIONS
 Recognizing: M-IP.413
 Creating: I-M-IP.413.977
 Specializing: I-M-IP.413.977
 Recognizing: I-M-TERMINATOR.323
 (THE CONDITIONS * ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN
 THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
 ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES
 PERIOD) = M-BYPASS-NODE
 Reading ASSOCIATED
 Recognizing: M-IP.413
 Creating: I-M-IP.413.977
 Specializing: I-M-IP.413.977
 Recognizing: I-M-TERMINATOR.323
 (THE CONDITIONS * ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN
 THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
 ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES
 PERIOD) = M-BYPASS-NODE
 Reading WITH
 (THE CONDITIONS ASSOCIATED WITH * THE REPLACEMENT OF A RC WITHIN
 THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
 ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES
 PERIOD) = M-BYPASS-NODE
 Reading REPLACEMENT
 (THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT * OF A RC WITHIN
 THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
 ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES
 PERIOD) = M-BYPASS-NODE
 Reading OF
 (THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF * A RC WITHIN
 THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
 ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES

PERIOD) = M-BYPASS-NODE
 Reading A
 Recognizing: M-IP.395
 Creating: I-M-IP.395.980
 Specializing: I-M-IP.395.980
 Recognizing: I-M-TERMINATOR.323
 (*LEFT-PAREN* A *RIGHT-PAREN*) = M-ENUMERATION
 (THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A * RC WITHIN
 THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
 ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES
 PERIOD) = M-BYPASS-NODE
 Reading RC
 Recognizing: M-IP.399
 Creating: I-M-IP.399.981
 Specializing: I-M-IP.399.981
 Recognizing: I-M-TERMINATOR.323
 (RC *) = M-RC referenced
 Recognizing: M-RC
 Removing: I-M-RC.982
 Specializing: I-M-RC.460
 Removing: I-M-RC.460
 (THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC * WITHIN
 THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
 ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES
 PERIOD) = M-BYPASS-NODE
 Reading WITHIN
 (THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN *
 THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY
 IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) =
 M-BYPASS-NODE
 Reading THE
 (THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE
 * NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY
 IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) =
 M-BYPASS-NODE
 Reading NETWORK
 Recognizing: M-IP.347
 Creating: I-M-IP.347.988
 Specializing: I-M-IP.347.988
 Recognizing: I-M-TERMINATOR.323
 (DMS NETWORK *) = M-CORE-NETWORK referenced
 Recognizing: M-CORE-NETWORK
 Removing: I-M-CORE-NETWORK.989
 Specializing: I-M-CORE-NETWORK.989
 Removing: I-M-IP.347.988
 (THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE
 NETWORK * CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY
 IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) =
 M-BYPASS-NODE
 Reading ARE
 (THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE
 NETWORK CONFIGURATION * ARE SUCH THAT THE NETWORK IS ALREADY
 IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) =
 M-BYPASS-NODE
 Reading ARE

IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading SUCH
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH * THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading THAT
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT * THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading THE
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE * NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading NETWORK
Recognizing: M-IP.347
Creating: I-M-IP.347.992
Specializing: I-M-IP.347.992
Recognizing: I-M-TERMINATOR.323
(DMS NETWORK *) = M-CORE-NETWORK referenced

Recognizing: M-CORE-NETWORK
Removing: I-M-CORE-NETWORK.993
Specifying: I-M-CORE-NETWORK.471
Removing: I-M-IP.347.992

(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK * IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading IS
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS * ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading ALREADY
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY * * IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading IN
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY * * IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading A
Recognizing: M-IP.395
Creating: I-M-IP.395.994
Specializing: I-M-IP.395.994
Recognizing: I-M-TERMINATOR.323
(*LEFT-PAREN* A * *RIGHT-PAREN*) = M ENUMERATION
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY * * IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading RECONFIGURED
Recognizing: M-IP.402
Creating: I-M-IP.402.995
Specializing: I-M-IP.402.995
Recognizing: I-M-TERMINATOR.323
((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED * STATE WITH THE FAILED (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED * STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading STATE
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE * SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE
((RESULT I-M-OPERATIONAL.670) IN A RECONFIGURED STATE * WITH THE FAILED (FAILED-COMPONENT M-ORU) BYPASSED) = M-BYPASS-NODE

Reading SUPPLYING
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading FULL
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading SERVICES
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE

Reading *PERIOD*
(THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*) = M-BYPASS-NODE referenced

Recognizing: M-BYPASS-NODE
Creating: I-M-BYPASS-NODE
Specializing: I-M-BYPASS-NODE.996
Specifying: I-M-BYPASS-NODE.996
Recognizing: M-BIPASS-NODE
Creating: I-M-BIPASS-NODE
Specializing: I-M-BIPASS-NODE.404
Specifying: I-M-BIPASS-NODE.404
Recognizing: M-HL-PATTERN.404
Creating: I-M-HL-PATTERN.404.997
Specializing: I-M-HL-PATTERN.404.997
(SEQUENCE-OF-STEPS I-M-FAILURE-EFFECT.970)
(JUSTIFICATION I-M-BYPASS-NODE.996 *) = M-SEARCH-FOR-EFFECT-REFERENT referenced

Recognizing: M-SEARCH-FOR-EFFECT-REFERENT
Creating: I-M-SEARCH-FOR-EFFECT-REFERENT.999
Specializing: I-M-SEARCH-FOR-EFFECT-REFERENT.999
Specifying: I-M-IP.412.971
Recognizing: M-HL-PATTERN.416
Creating: I-M-HL-PATTERN.416.1000
Specializing: I-M-HL-PATTERN.416.1000
Specifying: I-M-IP.413.977
Removing: I-M-IP.402.995
Specifying: I-M-IP.395.994
Removing: I-M-IP.395.994
Specifying: I-M-IP.395.994

Reading *LEFT-PAREN*

Reading B

Recognizing: M-IP.396
Creating: I-M-IP.396.1.001
Specializing: I-M-IP.396.1.001
Recognizing: I-M-TERMINATOR.323
(*LEFT-PAREN* B * *RIGHT-PAREN*) = M-ENUMERATION referenced

Reading *RIGHT-PAREN*

(*LEFT-PAREN* B *RIGHT-PAREN*) = M-ENUMERATION referenced

Recognizing: M-ENUMERATION
Creating: I-M-ENUMERATION.1002
Removing: I-M-ENUMERATION.1002
Specializing: I-M-ENUMERATION.605
Recognizing: M-BUILD-CONTEXT
Removing: I-M-BUILD-CONTEXT.1003
Specializing: M-BUILD-CONTEXT.959
Recognizing: M-CONTEXT
Removing: I-M-CONTEXT.961
Specializing: I-M-BUILD-CONTEXT.961
Removing: I-M-HL-PATTERN.416.1000
Removing: I-M-HL-PATTERN.416.973
Recognizing: M-BUILD-PROCEDURE
Creating: I-M-BUILD-PROCEDURE.1005
Specializing: I-M-BUILD-PROCEDURE.1005
Removing: I-M-HL-PATTERN.404.997
Recognizing: M-UNIFY-CORRECTION-MOPS
Creating: I-M-UNIFY-CORRECTION-MOPS.1006
Specializing: I-M-CONTEXT.1007
Specifying: I-M-CONTEXT.1007
Creating: I-M-CONTEXT.1007
Removing: I-M-IP.396.1.001

Reading MISSION

Recognizing: M-IP.341
Creating: I-M-IP.341.1008
Specializing: I-M-IP.341.1008
(MISSION * SUPPORT) = M-MISSION-SUPPORT referenced

Reading SUPPORT

(MISSION SUPPORT *) = M-MISSION-SUPPORT referenced

Recognizing: M-MISSION-SUPPORT
Creating: I-M-MISSION-SUPPORT.1009
Specializing: I-M-MISSION-SUPPORT.1009
Recognizing: M-IP.415
Creating: I-M-IP.415.1010
Specializing: I-M-IP.415.1010
Recognizing: I-M-TERMINATOR.323
Creating: I-M-TERMINATOR.323
Specializing: I-M-TERMINATOR.323
(*EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
(SEQUENCE-OF-STEP M-PROCEDURE)) = M-FAILURE-EFFECT
Removing: I-M-IP.341.1008

Reading *COLON*

(*EFFECTED-COMPONENT I-M-MISSION-SUPPORT.1009) *COLON* *
(SEQUENCE-OF-STEP M-PROCEDURE)) = M-FAILURE-EFFECT

Reading NONE

Recognizing: M-IP.414
Creating: I-M-IP.414.1011
Specializing: I-M-IP.414.1011
Recognizing: I-M-TERMINATOR.323
(NONE * *PERIOD*) = M-DISAMBIGUATE-NONE

Reading *PERIOD*

(NONE *PERIOD* *) = M-DISAMBIGUATE-NONE referenced

Recognizing: M-DISAMBIGUATE-NONE
Removing: I-M-DISAMBIGUATE-NONE.1012
Specializing: I-M-NULL-PROCEDURE
Recognizing: I-M-NULL-PROCEDURE
(*EFFECTED-COMPONENT I-M-MISSION-SUPPORT.1009) *COLON*
(SEQUENCE-OF-STEP I-M-NULL-PROCEDURE *) = M-FAILURE-EFFECT referenced

Recognizing: M-FAILURE-EFFECT
Creating: I-M-FAILURE-EFFECT.1013
Specializing: I-M-FAILURE-EFFECT.1013
Recognizing: M-IP.412
Creating: I-M-IP.412.1014
Specializing: I-M-IP.412.1014
Recognizing: I-M-TERMINATOR.411
(SEQUENCE-OF-STEP M-FAILURE-EFFECT)
(JUSTIFICATION M-FAILURE-CORRECTION) = M-SEARCH-FOR-EFFECT-REFERENT

Recognizing: M-HL-PATTERN.387
Creating: I-M-HL-PATTERN.387.1015
Specializing: I-M-HL-PATTERN.387.1015
Recognizing: M-HL-PATTERN.416
Creating: I-M-HL-PATTERN.416.1016
Specializing: I-M-HL-PATTERN.416.1016
Removing: I-M-IP.415.1010
Removing: I-M-IP.414.1011

Reading *LEFT-PAREN*

Reading C

Recognizing: M-IP.397
Creating: I-M-IP.397.1018
Specializing: I-M-TERMINATOR.323
(*LEFT-PAREN* C * *RIGHT-PAREN*) = M-ENUMERATION referenced

Reading *RIGHT-PAREN*

(*LEFT-PAREN* C *RIGHT-PAREN*) = M-ENUMERATION referenced

Recognizing: M-IP.397
Creating: I-M-IP.397.1018
Specializing: I-M-TERMINATOR.323
(*LEFT-PAREN* C * *RIGHT-PAREN*) = M-ENUMERATION referenced

Reading SYSTEM

Recognizing: M-IP.342
Creating: I-M-IP.342.1024
Specializing: I-M-IP.342.1024
Recognizing: I-M-TERMINATOR.323
(SYSTEM *) = M-SYSTEMS referenced
Recognizing: M-SYSTEMS
Removing: I-M-SYSTEMS.1025
Specializing: I-M-SYSTEMS.1025

Recognizing: M-IP.415
 Creating: I-M-IP.415.1026
 Specializing: I-M-IP.415.1026
 Recognizing: I-M-TERMINATOR.323
 ((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
 ((SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT
 Removing: I-M-IP.312.1024
 Recognizing: M-IP.338
 Creating: I-M-IP.338.1027
 Specializing: I-M-IP.338.1027
 Recognizing: I-M-TERMINATOR.323
 (SYSTEM * MANAGEMENT) = M-SM

Reading *COLON*
 ((EFFECTED-COMPONENT I-M-SYSTEMS.480) *COLON* *
 ((SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading NONE
 Recognizing: M-IP.414
 Creating: I-M-IP.414.1028
 Specializing: I-M-IP.414.1028
 Recognizing: I-M-TERMINATOR.323
 ((NONE * *PERIOD*) - M-DISAMBIGUATE-NONE)

Reading *PERIOD*
 ((NONE * PERIOD*)) = M-DISAMBIGUATE-NONE referenced
 Recognizing: M-DISAMBIGUATE-NONE
 Removing: I-M-DISAMBIGUATE-NONE.1029
 Specializing: I-M-DISAMBIGUATE-NONE.969
 Recognizing: I-M-NULL-PROCEDURE
 ((EFFECTED-COMPONENT I-M-SYSTEMS.480) *COLON*
 ((SEQUENCE-OF-STEPS I-M-NULL-PROCEDURE)) = M-FAILURE-EFFECT
 referenced
 Recognizing: M-FAILURE-EFFECT
 Creating: I-M-FAILURE-EFFECT.1030
 Specializing: I-M-FAILURE-EFFECT.1030
 Recognizing: M-IP.412
 Specializing: I-M-IP.412.1031
 Recognizing: I-M-TERMINATOR.411
 ((SEQUENCE-OF-STEPS M-FAILURE-EFFECT) *
 (JUSTIFICATION M-FAILURE-CORRECTION)) = M-SEARCH-FOR-EFFECT-REFERENT
 Recognizing: M-HL-PATTERN.387
 Creating: I-M-HL-PATTERN.387.1032
 Specializing: I-M-HL-PATTERN.387.1032
 Recognizing: M-HL-PATTERN.416
 Creating: I-M-HL-PATTERN.416.1033
 Specializing: I-M-HL-PATTERN.416.1033
 Removing: I-M-IP.415.1026
 Removing: I-M-IP.414.1028
 Removing: I-M-IP.338.1027

Reading *LEFT-PAREN*
 Reading D
 Recognizing: M-IP.398
 Creating: I-M-IP.398.1035
 Specializing: I-M-IP.398.1035
 (*LEFT-PAREN* D *RIGHT-PAREN*) = M ENUMERATION

Reading *RIGHT-PAREN*
 (*LEFT-PAREN* D *RIGHT-PAREN*) = M ENUMERATION referenced
 Recognizing: M-ENUMERATION

Removing: I-M-ENUMERATION.1016
 Specializing: I-M-ENUMERATION.605
 Recognizing: M-BUILD-CONTEXT
 Removing: I-M-BUILD-CONTEXT.1037
 Specializing: I-M-BUILD-CONTEXT.959
 Recognizing: M-CONTEXT
 Removing: I-M-CONTEXT.1023
 Removing: I-M-HL-PATTERN.416.1033
 Recognizing: M-BUILD-PROCEDURE
 Creating: I-M-BUILD-PROCEDURE.1039
 Specializing: I-M-BUILD-PROCEDURE.1039
 Creating: I-M-CONTEXT.1040
 Specializing: I-M-CONTEXT.1040
 Removing: I-M-IP.398.1035
 Removing: I-M-IP.412.1031

Reading INTERFACES
 Recognizing: M-IP.343
 Creating: I-M-IP.343.1041
 Specializing: I-M-IP.343.1041
 Recognizing: I-M-INTERFACES.323
 ((INTERFACES *)) = M-INTERFACES referenced
 Recognizing: M-INTERFACES
 Creating: I-M-INTERFACES.1042
 Recognizing: M-IP.415
 Creating: I-M-IP.415.1043
 Specializing: I-M-IP.415.1043
 Recognizing: I-M-TERMINATOR.323
 ((EFFECTED-COMPONENT M-MISSION-COMPONENT) * *COLON*
 ((SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT
 Removing: I-M-IP.343.1041

Reading *COLON*
 ((EFFECTED-COMPONENT I-M-INTERFACES.1042) *COLON* *
 ((SEQUENCE-OF-STEPS M-PROCEDURE)) = M-FAILURE-EFFECT

Reading NONE
 Recognizing: M-IP.414
 Creating: I-M-IP.414.1044
 Specializing: I-M-IP.414.1044
 Recognizing: I-M-TERMINATOR.323
 ((NONE * *PERIOD*)) = M-DISAMBIGUATE-NONE

Reading *PERIOD*
 ((NONE * PERIOD*)) = M-DISAMBIGUATE-NONE referenced
 Recognizing: M-DISAMBIGUATE-NONE
 Removing: I-M-DISAMBIGUATE-NONE.1045
 Specializing: I-M-NULL-PROCEDURE
 ((EFFECTED-COMPONENT I-M-INTERFACES.1042) *COLON*
 ((SEQUENCE-OF-STEPS I-M-NULL-PROCEDURE)) = M-FAILURE-EFFECT
 referenced
 Recognizing: M-FAILURE-EFFECT
 Creating: I-M-FAILURE-EFFECT.1046
 Specializing: I-M-FAILURE-EFFECT.1046
 Recognizing: M-IP.412
 Creating: I-M-IP.412.1047
 Specializing: I-M-TERMINATOR.411
 ((SEQUENCE-OF-STEPS M-FAILURE-EFFECT) *
 (JUSTIFICATION M-FAILURE-CORRECTION)) = M-SEARCH-FOR-EFFECT-REFERENT
 Recognizing: M-HL-PATTERN.387
 Creating: I-M-HL-PATTERN.387.1048

Specializing: I-M-HL-PATTERN.387.1048
 Recognizing: M-HL-PATTERN.416
 Creating: I-M-HL-PATTERN.416.1049
 Specializing: I-M-HL-PATTERN.416.1049
 Removing: I-M-IP.415.1043
 Removing: I-M-IP.414.1044
 Reading *EOC*
 Recognizing: M-CONTEXT
 Removing: I-M-CONTEXT.1040
 Creating: I-M-CONTEXT.1051
 Specializing: I-M-CONTEXT.1051
 Removing: I-M-HL-PATTERN.416.1049
 Recognizing: M-BUILD-PROCEDURE
 Creating: I-M-BUILD-PROCEDURE.1052
 Specializing: I-M-BUILD-PROCEDURE.1052
 Removing: I-M-HL-PATTERN.387.1048
 Removing: I-M-HL-PATTERN.387.1032
 Removing: I-M-HL-PATTERN.387.1015
 Removing: I-M-HL-PATTERN.387.972
 Removing: I-M-HL-PATTERN.387.939
 Removing: I-M-HL-PATTERN.387.694
 Removing: I-M-HL-PATTERN.387.600
 Removing: I-M-HL-PATTERN.387.450
 Removing: I-M-HL-PATTERN.387.447
 Removing: I-M-HL-PATTERN.387.444
 Removing: I-M-HL-PATTERN.387.441
 Removing: I-M-HL-PATTERN.387.438
 Removing: I-M-HL-PATTERN.387.431
 Removing: I-M-HL-PATTERN.387.423
 Recognizing: M-CASE
 Creating: I-M-CASE.1052
 Specializing: I-M-CASE.1053
 Removing: I-M-IP.403.451
 Removing: I-M-IP.412.1047

ITEM NAME *COLON* RING CONCENTRATOR FAILURE MODE *COLON* LOSS OF
 OUTPUT - FAILURE TO START FAILURE CAUSES *COLON* PIECE-PART
 FAILURES *COLON* CONTAMINATION *COLON* TEMPERATURE *LEFT-PAREN*
 HIGH OR LOW *RIGHT-PAREN* *COLON* MECHANICAL SHOCK *COLON*
 THERMAL SHOCK FAILURE DETECTION *SLASH* VERIFICATION *COLON*
 INDICATION OF A RC FAILURE IS FIRST DETECTED BY THE *QUOTE* NEXT
 QUOTE ACTIVE NODE ON THE NETWORK *PERIOD* SYSTEM MANAGEMENT
 WILL REACH A TIME-OUT LIMIT FOR RECEIPT OF THE NETWORK TOKEN
 PERIOD CORRECTIVE ACTION *COLON* LEFT-PAREN* A *RIGHT-PAREN*
 SHORT TERM *COLON* NETWORK RECONFIGURATION IS EFFECTED
 AUTOMATICALLY *PERIOD* THE DMS NETWORK REMAINS IN OPERATION IN A
 RECONFIGURED STATE WITH THE FAILED RC BYPASSED *PERIOD*
 LEFT-PAREN B *RIGHT-PAREN* LONG TERM *COLOR* THE CREWMEN CHECK
 FOR *QUOTE* APPLIED POWER *QUOTE* AND THE CREWMEN CHECK FOR
 QUOTE CONNECTOR TIGHTNESS *QUOTE* *PERIOD* IF THE RC CANNOT
 THEN BE PLACED IN OPERATION *COLON* THE RC IS REMOVED AND
 REPLACED WITH AN ORU LOGISTICS SPARE *PERIOD* FAILURE EFFECT ON
 COLON *LEFT-PAREN* A *RIGHT-PAREN* CREW *SLASH* SSPE *COLON*
 NONE *PERIOD* THE CONDITIONS ASSOCIATED WITH THE REPLACEMENT OF A
 RC WITHIN THE NETWORK CONFIGURATION ARE SUCH THAT THE NETWORK IS
 ALREADY IN A RECONFIGURED STATE SUPPLYING FULL SERVICES *PERIOD*
 LEFT-PAREN B *RIGHT-PAREN* MISSION SUPPORT *COLON* NONE
 PERIOD *LEFT-PAREN* C *RIGHT-PAREN* SYSTEM *COLON* NONE
 PERIOD *LEFT-PAREN* D *RIGHT-PAREN* INTERFACES *COLON* NONE
 PERIOD *EOF*)

(I-M-CASE.1053 (FAILED-COMPONENT I-M-GENERIC-RC.422)
 (MODE I-M-STARTUP-NO-OUTPUT.1089
 (SEQUENCE-OF-STEPS I-M-AND-GROUP .97.1085
 (2 I-M-WAIT-TO-RECEIVE-EVENT .95.1071
 (RESULT I-M-TIME-ELAPSED-FOR-OBJECT.25.1056
 (OBJECT I-M-SYSTEM-SM))
 (RESULT I-M-READY-TO-RECEIVE.22.1059
 (RECIPIENT I-M-SYSTEM-SM))
 (ACTION I-M-WAIT-FOR-MESSAGE.23.1064
 (OBJECT I-M-TOKEN.491))
 (PRECOND I-M-READY-TO-RECEIVE.22.1059
 (ACTOR I-M-SYSTEM-SM) (RECIPIENT I-M-SYSTEM-SM))
 (OBJECT I-M-TOKEN.491))
 (RECIPIENT I-M-SYSTEM-SM)) (MESSAGE I-M-TOKEN.491))
 (3 I-M-TIME-OUT-EVENT.96.1084
 (RESULT I-M-TRANSMISSION-TIMED-OUT.41.502
 (RECIPIENT I-M-SYSTEM-SM))
 (OBJECT I-M-TOKEN.491))
 (ACTION I-M-TIME-OUT.40.499 (ACTOR I-M-SYSTEM-SM))
 (RECIPIENT I-M-SYSTEM-SM)) (MESSAGE I-M-TOKEN.491))
 (PRECOND I-M-READY-TO-RECEIVE.93.1088
 (RECIPIENT I-M-SYSTEM-SM) (OBJECT I-M-TOKEN.491))
 (CAUSE I-M-FAULTY-COMPONENT-CAUSE.1094
 (CAUSE I-M-CONTAMINATION-CAUSE.440)
 (CAUSE I-M-TEMPERATURE-OUT-OF-RANGE-CAUSE.443)
 (CAUSE I-M-MECHANICAL-SHOCK-CAUSE.446)
 (CAUSE I-M-THERMAL-SHOCK-CAUSE.449)
 (DETECTION I-M-NON-OPERATIONAL.148.1100
 (RESULT I-M-DETECTION.1150
 (OBJECT I-M-GENERIC-RC.422))
 (PRECOND I-M-POWER-ON.141.1102 (OBJECT I-M-NEXT-RC.472))
 (PRECOND I-M-NEXT-NODE.140.1105 (OBJECT1 I-M-NEXT-RC.472))
 (OBJECT2 I-M-GENERIC-RC.422))
 (PRECOND I-M-LOGICALLY-CONNECTED.137.1107
 (OBJECT1 I-M-GENERIC-RC.422))
 (PRECOND I-M-PHYSICALLY-CONNECTED.136.1109
 (OBJECT1 I-M-GENERIC-RC.422))
 (FAILED-COMPONENT I-M-GENERIC-RC.422)
 (DETECTION-COMPONENT I-M-NEXT-RC.472)
 (SEQUENCE-OF-STEPS I-M-DETECTION-AND-GROUP.147.1147
 (1 I-M-DETECTION-PROC.146.1144
 (PRECOND I-M-READY-TO-RECEIVE.142.563
 (RECIPIENT I-M-TOKEN.491))
 (SEQUENCE-OF-STEPS I-M-AND-GROUP.145.1142
 (1 I-M-WAIT-TO-RECEIVE-EVENT.143.1123
 (RESULT I-M-TIME-ELAPSED-FOR-OBJECT.25.1056
 (OBJECT I-M-SYSTEM-SM))
 (RESULT I-M-READY-TO-RECEIVE.22.536
 (OBJECT I-M-TOKEN.491))
 (ACTION I-M-WAIT-FOR-MESSAGE.23.538
 (OBJECT I-M-TOKEN.491))
 (PRECOND I-M-READY-TO-RECEIVE.22.536
 (RECIPIENT I-M-TOKEN.491))
 (MESSAGE I-M-TOKEN.491))
 (2 I-M-TIME-OUT-EVENT.144.507
 (PRECOND I-M-READY-TO-RECEIVE.36.495
 (RECIPIENT I-M-SYSTEM-SM))

>
(dribble)

(SEQUENCE-OF-STEPS I-M-PROCEDURE.1272
(SEQUENCE-OF-STEPS I-M-NULL-AND-GROUP.86
{1 I-M-NULL-EVENT.85 (ACTION I-M-NULL-ACTION)}))

* FANSYS: A Computer Model of Text Comprehension and Question
 * Answering for Failure Analysis
 * File: qa_trace
 * Developed by: Sergio J. Alvarado
 Rona J. K. Braun
 Kenrick J. Mock
 * Artificial Intelligence Laboratory
 Computer Science Department
 University of California
 Davis, CA 95616
 * Funds for the support of this study have been allocated by the
 NASA-Ames Research Center, Moffett Field, California, under
 Interchange No. NCA2-721.
 * (start-shell)
 Starting the question shell...

> what is the failure cause for the time generation unit ?
 (WHAT IS THE FAILURE CAUSE FOR THE TIME GENERATION UNIT *Q-MARK*)
 Parsing (WHAT IS THE FAILURE CAUSE FOR THE TIME GENERATION UNIT *Q-MARK*) .
 Reading WHAT
 Recognizing: M-IP.4385
 Creating: I-M-IP.4385.4410
 Specializing: I-M-IP.4385.4410
 Recognizing: I-M-TERMINATOR.437
 (WHAT * IS THE (REQUESTED M-QUESTION-CONSTRAINT) AND THE
 (REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
 WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
 Q-MARK) - M-QUESTION
 Recognizing: M-IP.4384
 Creating: I-M-IP.4384.4411
 Specializing: I-M-IP.4384.4411
 Recognizing: I-M-TERMINATOR.437
 (WHAT * ARE THE (REQUESTED M-AND-GROUP) IN THE (GIVEN M-QUESTION-CONSTRAINT)
 WHEN THE (STATE M-STATE-ASSERTION) *Q-MARK*) - M-QUESTION2
 Reading IS
 Recognizing: M-IP.4388
 Creating: I-M-IP.4388.4412
 Specializing: I-M-IP.4388.4412
 Removing: I-M-IP.4388.4412
 (WHAT IS * THE (REQUESTED M-QUESTION-CONSTRAINT) AND THE
 (REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
 WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
 Q-MARK) - M-QUESTION
 Reading THE
 Recognizing: M-IP.481
 Creating: I-M-IP.481.4413
 Specializing: I-M-IP.481.4413
 Recognizing: I-M-TERMINATOR.437
 (THE *
 (ACTOR M-ROOT)

PHYSICALLY
 CONNECTS
 THE
 (OBJECT1 M-ROOT)
 TO
 THE
 (OBJECT2 M-ROOT)
 PERIOD = M-PHYSICAL-CONNECT-EVENT
 Recognizing: M-IP.479
 Creating: I-M-IP.479.4414
 Specializing: I-M-IP.479.4414
 Recognizing: I-M-TERMINATOR.437
 (THE *
 (ACTOR M-ROOT)
 LOGICALLY
 CONNECTS
 THE
 (OBJECT1 M-ROOT)
 TO
 THE
 (OBJECT2 M-ROOT)
 PERIOD = M-LOGICAL-CONNECT-EVENT
 (WHAT IS THE * (REQUESTED M-QUESTION-CONSTRAINT) AND THE
 (REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
 WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
 Q-MARK) - M-QUESTION
 Reading FAILURE
 Reading CAUSE
 Recognizing: M-IP.4395
 Creating: I-M-IP.4395.4415
 Specializing: I-M-IP.4395.4415
 Recognizing: I-M-TERMINATOR.437
 (FAILURE CAUSE *) - M-FAILURE-CAUSE referenced
 Recognizing: M-FAILURE-CAUSE
 Creating: I-M-FAILURE-CAUSE.4416
 Specializing: I-M-FAILURE-CAUSE.4416
 (WHAT IS THE (REQUESTED M-QUESTION-CONSTRAINT) * AND THE
 (REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
 WHEN THE (STATE M-STATE-ASSERTION) FOR THE (GIVEN M-QUESTION-CONSTRAINT)
 WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
 Q-MARK) - M-QUESTION
 Removing: I-M-IP.4395.4415
 Reading FOR
 (WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.4416) AND THE
 (REQUESTED M-QUESTION-CONSTRAINT) FOR * THE (GIVEN M-QUESTION-CONSTRAINT)
 WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
 Q-MARK) - M-QUESTION
 Reading THE
 Recognizing: M-IP.481
 Creating: I-M-IP.481.4417
 Specializing: I-M-IP.481.4417
 Recognizing: I-M-TERMINATOR.437
 (THE *
 (ACTOR M-ROOT)
 PHYSICALLY
 CONNECTS
 THE
 (OBJECT1 M-ROOT)
 TO
 THE
 (OBJECT2 M-ROOT)

PERIOD = M-PHYSICAL-CONNECT-EVENT
 Recognizing: M-IP.419
 Creating: I-M-IP.479.4418
 Specializing: I-M-IP.479.4418
 Recognizing: I-M-TERMINATOR.437
 (THE *
 (ACTOR M-ROOT)
 LOGICALLY
 CONNECTS
 THE
 (OBJECT1 M-ROOT)
 TO
 THE
 (OBJECT2 M-ROOT)
 PERIOD = M-LOGICAL-CONNECT-EVENT
 (WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.4416) AND THE
 (REQUESTED M-QUESTION-CONSTRAINT) FOR THE * (GIVEN M-QUESTION-CONSTRAINT)
 WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
 Q-MARK) = M-QUESTION

Reading TIME

Recognizing: M-IP.448

Creating: I-M-IP.448.4419

Specializing: I-M-IP.448.4419

Recognizing: I-M-TERMINATOR.437

(TIME * GENERATION UNIT) = M-GENERIC-TGU

Reading GENERATION

(TIME GENERATION * UNIT) = M-GENERIC-TGU

Reading UNIT

(TIME GENERATION UNIT *) = M-GENERIC-TGU referenced

Recognizing: M-GENERIC-TGU.4420

Removing: I-M-GENERIC-TGU.4420

Specializing: I-M-GENERIC-TGU.3723
 (WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.4416) AND THE
 (REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN M-QUESTION-CONSTRAINT) *
 WHEN THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION)
 Q-MARK) = M-QUESTION

Removing: I-M-IP.448.4419

Reading *Q-MARK*

(WHAT IS THE (REQUESTED I-M-FAILURE-CAUSE.4416) AND THE
 (REQUESTED M-QUESTION-CONSTRAINT) FOR THE (GIVEN I-M-GENERIC-TGU.3723) WHEN
 THE (STATE M-STATE-ASSERTION) AND THE (STATE M-STATE-ASSERTION) *Q-MARK*) = M-QUEST
 TION referenced

Recognizing: M-QUESTION

Creating: I-M-QUESTION.4421

Specializing: I-M-QUESTION.4421

In: LAMBDA (MOPNAME)
 (LET ((GIVEN #) (REQUESTED #) (STATES #) (RESULTS NIL))
 (FOR (STATE :IN STATES) :DO (SETF GIVEN #))
 (QA GIVEN REQUESTED))

Warning: Variable RESULTS defined but never used.

(LET ((MOP #) (COUNTER 0))

(COND (#))

Warning: Variable COUNTER defined but never used.

Searching bottom-up for Entry-MOP based on requested:

M-CASE

Performing Direct Search.

Performing Direct Search. Indices: ((FAILED-COMPONENT M-GENERIC-TGU))

Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE

Input matches indices to: (COMP.4199)

Current MOP: COMP.4199

Found: NIL

Values: NIL

Picking out those that match 100*

Results of Direct Search:

Results of Direct Search:

NIL

Elaborating using requested information.

Elaborating using requested information.

* matches with anything. Indices: ((FAILED-COMPONENT M-GENERIC-TGU))

(CAUSE *) (CAUSE *)

Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE

Input matches indices to: (COMP.4376 COMP.1326 COMP.1325 COMP.1324 COMP.1323
 COMP.1322 COMP.4199)

Performing Search. Entry MOP=M-CASE

Current MOP: COMP.4376

Input matches indices to: (I-M-GW.3 I-M-TGU.3)
 Current MOP: I-M-GW.3
 Current MOP: I-M-TGU.3
 Current MOP: COMP.1326
 Input matches indices to: (I-M-RC.2)

Current MOP: COMP.4376

Input matches indices to: (I-M-RC.2)
 Current MOP: I-M-RC.2
 Current MOP: COMP.1325
 Input matches indices to: (I-M-RC.2)

Current MOP: COMP.4376

Input matches indices to: (I-M-RC.2)
 Current MOP: I-M-RC.2
 Current MOP: COMP.1325
 Input matches indices to: (I-M-RC.2)

Current MOP: COMP.4376

Input matches indices to: (I-M-RC.2)
 Current MOP: I-M-RC.2
 Current MOP: COMP.4199
 Input matches indices to: (I-M-TGU.2)

Current MOP: I-M-TGU.2

Found: (I-M-GW.3 COMP.4376 I-M-TGU.3 COMP.1326 COMP.1325 COMP.1324 COMP.1323
 COMP.1322 I-M-RC.2 M-CASE COMP.4199 I-M-TGU.2)

Values: ((I-M-GW.3 2/3) (COMP.4376 0) (I-M-TGU.3.1) (COMP.1325 0)
 (COMP.1324 0) (COMP.1323 0) (COMP.1322 0) (I-M-RC.2 20/27) (M-CASE 0)
 (COMP.4199 0) (I-M-TGU.2 1))

Picking out those that match 100%

Results of elaboration:
Results of elaboration:

(I-M-TGU.3 I-M-TGU.2)

- [1] ERRONEOUS INPUT
- [2] PIECE-PART FAILURES
- [3] CONTAMINATION
- [4] TEMPERATURE (HIGH OR LOW)
- [5] MECHANICAL SHOCK
- [6] THERMAL SHOCK

Found good match. Elaborate further anyway? (Y/n)

n
 Removing: I-M-IP.4385.4410
 Removing: I-M-IP.479.4418
 Removing: I-M-IP.481.4417
 Removing: I-M-IP.479.4414
 Removing: I-M-IP.481.4413
 Removing: I-M-IP.4384.4411

* (dribble)

```

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: generation_trace
Developed by: Sergio J. Alvarado
              Ronald K. Braun
              Kenrick J. Mock
Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616
Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.
Starts dribbling to gentrace (1993/6/16, 21:17:38).
NIL
> (show I-M-TIME-OUT-EVENT.144.507)
#S (MOP NAME I-M-TIME-OUT-EVENT.144.507 ABSTS (M-TIME-OUT-EVENT.144
ALL-ABSTS
(I-M-TIME-OUT-EVENT.144.507 M-TIME-OUT-EVENT.144
M-TIME-OUT-EVENT M-TRANSMISSION-EVENT M-EVENT-STEP
M-ROOT)
SPEC$ NIL SLOTS
  (PRCND I-M-READY-TO-RECEIVE.36.495)
  (ACTION I-M-TIME-OUT .40 .499)
  (RESULT I-M-TRANSMISSION-TIMED-OUT .41 .502)
  (RECIPIENT I-M-SYSTEM-SM) (MESSAGE I-M-TOKEN.491))
TYPE INSTANCE IPS NIL ASSOC-FNS NIL PMS NIL AMS
(*S (AM PARENT I-M-TIME-OUT-EVENT.144.507 OWNERS
  (M-PROCESSING-MOP M-HL-PATTERN M-HL-PATTERN.416
    I-HL-PATTERN PATTERN.416.527 M-EVENT-STEP M-EVENT
    M-TRANSMISSION-EVENT M-TIME-OUT-EVENT
    M-TIME-OUT-EVENT.144 I-M-TIME-OUT-EVENT.144.507)
  START 73 FINISH 73)
IP-REFS NIL VARS
((MESSAGE I-M-TOKEN.491) TOKEN.134 MESSAGE.35)) BACK
((2 I-M-AND-GROUP.145.1142) (2 I-M-AND-GROUP.145.534)
(2 I-M-AND-GROUP.145.534) (STEP I-M-BUILD-PROCEDURE.533))
INDEX-DOWN NIL INDEX-UP NIL NORMS NIL BAD-INDEX NIL ELABORATIONS
NIL EQUIVALENT (I-M-TIME-OUT-EVENT.144.507)
(I-M-TIME-OUT-EVENT.144.507)
> (generate I-M-TIME-OUT-EVENT.144.507)
  Checking mop I-M-TIME-OUT-EVENT.144.507 for phrases:
  Found: NIL
  Checking mop M-TIME-OUT-EVENT.144 for phrases:
  Found: NIL
  Checking mop M-TIME-OUT-EVENT for phrases:
  Found: (M-IP.358)
Considering phrase M-IP.358: ((1
  (M-SYNTAX-CATEGORY RECIPIENT M-SYSTEM)
  (2 WILL) (3 REACH) (4 A) (5 TIME-OUT)
  (6 LIMIT) (7 FOR) (8 RECEIPT) (9 OF)
  (10 THE)
  (11
    (M-SYNTAX-CATEGORY MESSAGE M-MESSAGE)))
  (TARGET M-TIME-OUT-EVENT)
  (TERMINATOR I-M-TERMINATOR.323)
  (STORE 5) (OPTIONALS NIL) (LENGTH 12))

Using phrase: M-IP.358
Checking mop I-M-SYSTEM-SM for phrases:
  Found: NIL
  Checking mop M-SM for phrases:
  Found: (M-IP.338)
Considering phrase M-IP.338: ((1 SYSTEM) (2 MANAGEMENT) (TARGET M-SM)
  (TERMINATOR I-M-TERMINATOR.323)
  (STORE 1) (OPTIONALS NIL) (LENGTH 2))

Using phrase: M-IP.338
Checking mop I-M-TOKEN.491 for phrases:
  Found: NIL
  Checking mop M-TOKEN for phrases:
  Found: (M-IP.339)
Considering phrase M-IP.339: ((1 NETWORK) (2 TOKEN) (TARGET M-TOKEN)
  (TERMINATOR I-M-TERMINATOR.323)
  (STORE 2) (OPTIONALS NIL) (LENGTH 2))

Using phrase: M-IP.339
  (*SYSTEM* "MANAGEMENT" "WILL" "REACH" "A" "TIME-OUT" "LIMIT" "FOR"
  "RECEIPT" "OF" "THE" "NETWORK" "TOKEN" ".")
  >
  (dribble)

```

[2] LONG TERM CORRECTION PROCEDURE : THE CREW PHYSICALLY DISCONNECTS THE GATEWAY FROM THE DMS NETWORK . THE CREW PHYSICALLY CONNECTS THE BACKUP GATEWAY TO THE DMS NETWORK . THE CREW LOGICALLY DISCONNECTS THE GATEWAY FROM THE DMS NETWORK . THE CREW LOGICALLY CONNECTS THE BACKUP GATEWAY TO THE DMS NETWORK .

Found good match. Elaborate anyway? (Y/n) n

(I-M-LOOKUP-BACKUP .2033 I-M-REPLACE-WITH-Spare .426.2106)

Failure Correction Procedure for the Standard Data Processor
When the Correction Procedure = Lookup-Backup?

* (qa ' (m-sdp m-lookup-backup) '(m-correction-and-group))

Searching bottom-up for Entry-MOP based on requested:

M-OR-GROUP

M-AND-GROUP

Searching bottom-up for Entry-MOP based on given:

Performing Direct Search. Indices: (NIL (FAILED-COMPONENT M-SDP))

Performing Search. Entry MOP=M-LOOKUP-BACKUP

Current MOP: M-LOOKUP-BACKUP
Input matches indices to: (I-M-LOOKUP-BACKUP .3061)

Current MOP: I-M-LOOKUP-BACKUP .3061

Found: (M-LOOKUP-BACKUP I-M-LOOKUP-BACKUP .3061)

Values: ((M-LOOKUP-BACKUP 0) (I-M-LOOKUP-BACKUP .3061 1))

Picking out those that match 100%

Performing Search. Entry MOP=M-LOOKUP-BACKUP

Current MOP: M-LOOKUP-BACKUP
Input matches indices to: (I-M-LOOKUP-BACKUP .3061)

Current MOP: I-M-LOOKUP-BACKUP .3061

Found: (M-LOOKUP-BACKUP I-M-LOOKUP-BACKUP .3061)

Values: ((M-LOOKUP-BACKUP 0) (I-M-LOOKUP-BACKUP .3061 1))

Picking out those that match 100%

Results of Direct Search:

(I-M-LOOKUP-BACKUP .3061 I-M-LOOKUP-BACKUP .3061)

[1] THE SYSTEM MANAGEMENT SELECTS A BACKUP STANDARD DATA PROCESSOR TO REPLACE THE STANDARD DATA PROCESSOR FROM A LOOK-UP TABLE MAINTAINED WITHIN THE DMS . THE SYSTEM MANAGEMENT POWERS DOWN THE STANDARD DATA PROCESSOR . THE SYSTEM MANAGEMENT

```

* * FANSYS: A Computer Model of Text Comprehension and Question
* * Answering for Failure Analysis
* * File: memory_retrieval_traces
* *
* * Developed by: Sergio J. Alvarado
* * Ronald K. Braun
* * Kendrick J. Mock
* *
* * Artificial Intelligence Laboratory
* * Computer Science Department
* * University of California
* * Davis, CA 95616
* *
* * Funds for the support of this study have been allocated by the
* * NASA-Ames Research Center, Moffett Field, California, under
* * Interchange No. NCA2-721.
* *
* * Memory Search and Retrieval Traces. Full traces from
* * section 6.
* *
* * Failure Correction Procedure for the Gateway when the Failure
* mode is Startup-No-Output?
* *
* (qa '(m-gw m-startup-no-output) '(m-failure-correction))
* Searching bottom-up for Entry-MOP based on requested:
* *
M-CASE
Performing Direct Search. Indices: ((MODE M-STARTUP-NO-OUTPUT)
(FAILED-COMPONENT M-GW))
Performing Search. Entry MOP=M-CASE
Current MOP: M-CASE
Input matches indices to: (COMP .2499)
Current MOP: COMP .2499
Input matches indices to: (I-M-GW.1)
Current MOP: I-M-GW.1
Found: (M-CASE COMP .2499 I-M-GW.1)
Values: ((M-CASE 0) (COMP .2499 0) (I-M-GW.1 1))
Picking out those that match 1004
Results of Direct Search:
(I-M-GW.1)

[1] SHORT TERM CORRECTION PROCEDURE : THE SYSTEM MANAGEMENT
SELECTS A BACKUP GATEWAY TO REPLACE THE GATEWAY FROM A LOOK-UP
TABLE MAINTAINED WITHIN THE DMS . THE SYSTEM MANAGEMENT
POWERS DOWN THE GATEWAY . THE SYSTEM MANAGEMENT POWERS UP
THE BACKUP GATEWAY . CORRECTION PROCEDURE : THE SYSTEM

```

POWERS UP THE BACKUP STANDARD DATA PROCESSOR . CORRECTION
 PROCEDURE : THE SYSTEM MANAGEMENT LOGICALLY DISCONNECTS THE STANDARD DATA PROCESSOR FROM THE DMS NETWORK . THE SYSTEM MANAGEMENT LOGICALLY CONNECTS THE BACKUP STANDARD DATA PROCESSOR TO THE DMS NETWORK .

(I-M-SDP.2 I-M-GW.2)

Found good match. Elaborate anyway? (Y/n) n
 Failed Component when the Detection Procedure = Heartbeat?

 * (qa '(m-heartbeat-detection) '(m-oru))
 Searching bottom-up for Entry-MOP based on requested:

M-CASE
 Performing Direct Search. Indices: ((DETECTION M-HEARTBEAT-DETECTION))
 Performing Direct Search. Indices: ((DETECTION M-HEARTBEAT-DETECTION))
 Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE
 Found: NIL
 Values: NIL

Picking out those that match 100%

Results of Direct Search:
 Results of Direct Search:
 NIL

Elaborating using requested information.
 Elaborating using requested information.

* matches with anything. Indices: ((DETECTION M-HEARTBEAT-DETECTION)
 (FAILED-COMPONENT *))

Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE
 Input matches indices to: (COMP.3569 COMP.2499 COMP.1321 COMP.4199)

Current MOP: COMP.3569
 Input matches indices to: (I-M-SDP.2)

Current MOP: I-M-SDP.2

Current MOP: COMP.2499
 Input matches indices to: (I-M-GW.2)

Current MOP: I-M-GW.2

Current MOP: COMP.1321
 Current MOP: COMP.4199

Found: (COMP.3569 I-M-SDP.2 M-CASE COMP.2499 I-M-GW.2)

Values: ((COMP.3569 0) (I-M-SDP.2 1) (M-CASE 0) (COMP.2499 0) (I-M-GW.2 1))

Picking out those that match 100%

Results of elaboration:

Found good match. Elaborate further anyway? (Y/n) n

Failure Cause when the mode is operational - erroneous output?

 * (qa '(m-operational-erroneous-output) '(m-failure-cause))

searching bottom-up for Entry-MOP based on requested:

M-CASE

Performing Direct Search.
 Performing Direct Search. Indices: ((MODE M-OPERATIONAL-ERRONEOUS-OUTPUT))

Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE
 Found: NIL

Values: NIL

Picking out those that match 100%

Results of Direct Search:

NIL

Elaborating using requested information.
 Elaborating using requested information.

* matches with anything. Indices: ((MODE M-OPERATIONAL-ERRONEOUS-OUTPUT)
 (CAUSE *))

Performing Search. Entry MOP=M-CASE

Current MOP: M-CASE
 Input matches indices to: (COMP.1326 COMP.1324 COMP.1323 COMP.1322
 COMP.4376)

Current MOP: COMP.1326

Input matches indices to: (I-M-RC.2)

Current MOP: I-M-RC.2

Current MOP: COMP.1325

Input matches indices to: (I-M-RC.2)

Current MOP: I-M-RC.2

Current MOP: COMP.1324

Input matches indices to: (I-M-RC.2)

Current MOP: I-M-RC.2

Current MOP: COMP.1323

Input matches Indices to: (I-M-RC.2)

Current MOP: I-M-RC.2

Current MOP: COMP.1322

Input matches Indices to: (I-M-RC.2)

Current MOP: I-M-RC.2

Current MOP: COMP.4376

Input matches Indices to: (I-M-GW.3 I-M-TGU.3)

Current MOP: I-M-GW.3

Current MOP: I-M-TGU.3

Found: (COMP.1326 COMP.1325 COMP.1324 COMP.1323 COMP.1322 I-M-RC.2 I-M-GW.3
M-CASE COMP.4376 I-M-TGU.3)

Values: ((COMP.1326 0) (COMP.1325 0) (COMP.1324 0) (COMP.1323 0) (COMP.1322 0)
(I-M-RC.2 141/196) (I-M-GW.3 1) (M-CASE 0) (COMP.4376 0) (I-M-TGU.3 1))

Picking out those that match 100%

Results of elaboration:

(1) M-GW.3 I-M-TGU.3)

[1] PIECE-PART FAILURES

[2] ERRONEOUS INPUT

Found good match. Elaborate further anyway? (Y/n) y

Elaborating on unspecified entry slots...

Indices: ((FAILED-COMPONENT *) (MODE M-OPERATIONAL-ERRONEOUS-OUTPUT)
(CAUSE *) (DETECTION *) (CORRECTION *) (EFFECT *))

Performing search. Entry MOP=M-CASE

Current MOP: M-CASE

Input matches Indices to: (COMP.3569 COMP.2499 COMP.1321 COMP.4376 COMP.1326
COMP.1325 COMP.1324 COMP.1323 COMP.1322 COMP.3575
COMP.3177 I-M-GW.3 COMP.1327 I-M-TGU.2 I-M-RC.1
I-M-SDP.3 COMP.2506 COMP.1781 COMP.1780 I-M-RC.1
I-M-TGU.3 COMP.4378 COMP.4377 COMP.4202 COMP.4201
COMP.3988 COMP.3987 COMP.3719 COMP.3718 COMP.3577
COMP.2576 COMP.3179 COMP.2792 COMP.2791
COMP.2508 COMP.2507 COMP.2135 COMP.2134 COMP.1783
COMP.1782 COMP.1330 COMP.1329 COMP.1328 COMP.4199)

Current MOP: COMP.3569

Input matches Indices to: (I-M-SDP.1 I-M-SDP.2 I-M-SDP.3)

Current MOP: I-M-SDP.1

Current MOP: I-M-SDP.2

Current MOP: I-M-SDP.3

Current MOP: COMP.2499

Input matches Indices to: (I-M-GW.2 I-M-GW.3)

Current MOP: I-M-GW.1

Current MOP: I-M-GW.2

Current MOP: I-M-GW.3

Current MOP: COMP.1321

Input matches Indices to: (COMP.3573 COMP.3175 I-M-TGU.2 I-M-GW.1 I-M-RC.3
COMP.1777 I-M-RC.1 I-H-RC.2)

Current MOP: COMP.3574

Input matches Indices to: (I-M-SDP.2 I-M-GW.2)

Current MOP: I-M-GW.3

Current MOP: I-M-TGU.3

Current MOP: COMP.3126

Input matches Indices to: (COMP.3574 COMP.3176 I-M-TGU.2 I-M-GW.1 I-M-SDP.2
I-M-SDP.1 I-M-GW.2 COMP.2505 I-M-GW.1 I-M-RC.3
COMP.1778 I-M-RC.1 I-M-RC.2)

Current MOP: COMP.3574

Input matches Indices to: (I-M-SDP.2 I-M-GW.2)

Current MOP: I-M-SDP.2

Current MOP: I-M-GW.2

Current MOP: COMP.3176

Input matches Indices to: (I-M-SDP.1 I-M-GW.1)

Current MOP: I-M-SDP.1

Current MOP: I-M-GW.1

Current MOP: I-M-TGU.2

Current MOP: I-M-TGU.1

Current MOP: I-M-SDP.2

Current MOP: I-M-SDP.1

Current MOP: I-M-GW.2

Current MOP: COMP.3177

Input matches Indices to: (COMP.3573 COMP.3175 I-M-TGU.2 I-M-GW.1 I-M-SDP.2
I-M-SDP.1 I-M-GW.2 COMP.2504 I-M-GW.1 I-M-RC.3
COMP.1777 I-M-RC.1 I-H-RC.2)

Current MOP: COMP.3573

Input matches Indices to: (I-M-SDP.2 I-M-GW.2)

Current MOP: I-M-SDP.2

Current MOP: I-M-GW.2

Current MOP: COMP.3175

Input matches Indices to: (I-M-SDP.1 I-M-GW.1)

Current MOP: I-M-SDP.1

Current MOP: I-M-GW.1

Current MOP: I-M-TGU.2

Current MOP: I-M-TGU.1

Current MOP: I-M-SDP.2

Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.2
 Current MOP: COMP.2504
 Input matches Indices to: (I-M-GW.2 I-M-SDP.2 I-M-SDP.1 I-M-GW.1)

Current MOP: I-M-GW.2
 Current MOP: I-M-SDP.2
 Current MOP: I-M-GW.1
 Current MOP: COMP.2502
 Input matches Indices to: (I-M-GW.2 I-M-SDP.2 I-M-SDP.1 I-M-GW.1)

Current MOP: I-M-GW.2
 Current MOP: I-M-SDP.2
 Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.1
 Current MOP: I-M-GW.1
 Current MOP: I-M-RC.1
 Current MOP: I-M-RC.3
 Current MOP: COMP.1775
 Current MOP: I-M-RC.1
 Current MOP: I-M-RC.2
 Current MOP: I-M-RC.2
 Current MOP: COMP.1324
 Input matches Indices to: (COMP.3572 COMP.3174 I-M-TGU.2 I-M-TGU.1 I-M-SDP.2
 I-M-SDP.1 I-M-GW.2 COMP.2503 I-M-GW.1 I-M-RC.3
 COMP.1776 I-M-RC.1 I-M-RC.2)

Current MOP: COMP.3572
 Input matches Indices to: (I-M-SDP.2 I-M-GW.2)

Current MOP: COMP.3570
 Input matches Indices to: (I-M-SDP.2 I-M-GW.2)

Current MOP: I-M-SDP.2
 Current MOP: I-M-GW.2
 Current MOP: COMP.3172
 Input matches Indices to: (I-M-SDP.1 I-M-GW.1)

Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.1
 Current MOP: I-M-TGU.3
 Current MOP: I-M-TGU.2
 Current MOP: I-M-TGU.1
 Current MOP: I-M-SDP.3
 Current MOP: I-M-SDP.2
 Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.3
 Current MOP: I-M-GW.2
 Current MOP: COMP.2501
 Current MOP: I-M-RC.1
 Current MOP: I-M-RC.2

Current MOP: COMP.3570
 Input matches Indices to: (COMP.3570 COMP.3172 I-M-TGU.3 I-M-TGU.2 I-M-TGU.1
 I-M-SDP.3 I-M-SDP.2 I-M-SDP.1 I-M-GW.3 I-M-GW.2
 COMP.2501 I-M-GW.1 I-M-RC.3 COMP.1774 I-M-RC.1
 I-M-RC.2)

Current MOP: COMP.3570
 Input matches Indices to: (I-M-SDP.2 I-M-GW.2)

Current MOP: I-M-SDP.2
 Current MOP: I-M-GW.2
 Current MOP: COMP.3172
 Input matches Indices to: (I-M-SDP.1 I-M-GW.1)

Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.1
 Current MOP: I-M-TGU.3
 Current MOP: I-M-TGU.2
 Current MOP: I-M-TGU.1
 Current MOP: I-M-SDP.3
 Current MOP: I-M-SDP.2
 Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.3
 Current MOP: I-M-GW.2
 Current MOP: COMP.2501
 Current MOP: I-M-RC.1
 Current MOP: I-M-RC.2

Current MOP: I-M-SDP.2
 Current MOP: I-M-GW.2
 Current MOP: COMP.3177
 Input matches Indices to: (I-M-SDP.1 I-M-GW.1)

Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.1
 Current MOP: I-M-GW.3
 Current MOP: COMP.1327
 Input matches Indices to: (I-M-RC.3 COMP.1779 I-M-RC.1 I-M-RC.2)

Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.1
 Current MOP: I-M-GW.2
 Current MOP: COMP.3173
 Input matches Indices to: (I-M-SDP.1 I-M-GW.1)

Current MOP: I-M-SDP.1
 Current MOP: I-M-GW.1
 Current MOP: I-M-GW.2
 Current MOP: I-M-GW.3
 Current MOP: COMP.1327
 Input matches Indices to: (I-M-RC.3 COMP.1779 I-M-RC.1 I-M-RC.2)

```

Current MOP: COMP.3718
Input matches indices to: (I-M-SDP.2 I-M-SDP.3)

Current MOP: I-M-RC.3
Current MOP: COMP.1779
Input matches indices to: (I-M-RC.2)

Current MOP: I-M-RC.2
Current MOP: COMP.1779
Input matches indices to: (I-M-RC.2)

Current MOP: I-M-RC.2
Current MOP: I-M-RC.1
Current MOP: I-M-TGU.2
Current MOP: I-M-TGU.1
Current MOP: I-M-SDP.3
Current MOP: COMP.2506
Input matches indices to: (I-M-GW.1 I-M-SDP.2 I-M-SDP.1 I-M-GW.3 I-M-GW.2)

Current MOP: I-M-GW.1
Current MOP: I-M-SDP.2
Current MOP: I-M-SDP.1
Current MOP: I-M-GW.3
Current MOP: I-M-GW.2
Current MOP: COMP.1781
Input matches indices to: (I-M-RC.3)

Current MOP: I-M-RC.3
Current MOP: COMP.1780
Input matches indices to: (I-M-RC.3)

Current MOP: I-M-RC.3
Current MOP: I-M-RC.1
Current MOP: I-M-TGU.3
Current MOP: COMP.4378
Input matches indices to: (I-M-TGU.2 I-M-TGU.3)

Current MOP: I-M-TGU.2
Current MOP: I-M-TGU.3
Current MOP: COMP.4202
Input matches indices to: (I-M-TGU.1 I-M-TGU.2)

Current MOP: I-M-TGU.1
Current MOP: I-M-TGU.2
Current MOP: COMP.4201
Input matches indices to: (I-M-TGU.1 I-M-TGU.2)

Current MOP: I-M-TGU.1
Current MOP: I-M-TGU.2
Current MOP: COMP.3988
Input matches indices to: (I-M-TGU.1 I-M-SDP.3)

Current MOP: I-M-TGU.1
Current MOP: I-M-SDP.3
Current MOP: COMP.3987
Input matches indices to: (I-M-TGU.1 I-M-SDP.3)

Current MOP: I-M-SDP.2
Current MOP: I-M-SDP.3

```

[GC threshold exceeded with 2,007,064 bytes in use. Commencing GC.]
 [GC completed with 29,656 bytes retained and 1,977,408 bytes freed.]
 [GC will next occur when at least 2,029,656 bytes are in use.]

```

Current MOP: COMP.3718
Input matches indices to: (I-M-SDP.1 I-M-GW.3)

Current MOP: I-M-SDP.2
Current MOP: COMP.3576
Input matches indices to: (I-M-SDP.2)

Current MOP: I-M-SDP.2
Current MOP: COMP.3179
Input matches indices to: (I-M-SDP.1 I-M-GW.3)

Current MOP: I-M-SDP.1
Current MOP: COMP.3178
Input matches indices to: (I-M-SDP.1 I-M-GW.3)

Current MOP: I-M-SDP.1
Current MOP: COMP.3179
Input matches indices to: (I-M-SDP.1 I-M-GW.3)

Current MOP: I-M-SDP.1
Current MOP: COMP.2792
Input matches indices to: (I-M-GW.2 I-M-GW.3)

Current MOP: I-M-SDP.1
Current MOP: COMP.2791
Input matches indices to: (I-M-GW.2 I-M-GW.3)

Current MOP: I-M-GW.2
Current MOP: COMP.2508
Input matches indices to: (I-M-GW.1 I-M-GW.2)

Current MOP: I-M-GW.1
Current MOP: COMP.2507
Input matches indices to: (I-M-GW.1 I-M-GW.2)

Current MOP: I-M-GW.1
Current MOP: COMP.2135
Input matches indices to: (I-M-GW.1 I-M-RC.3)

Current MOP: I-M-GW.1
Current MOP: I-M-RC.3
Current MOP: COMP.2134
Input matches indices to: (I-M-RC.3)

Current MOP: I-M-RC.3
Current MOP: COMP.1783
Input matches indices to: (I-M-RC.3)

Current MOP: I-M-RC.3
Current MOP: COMP.1782
Input matches indices to: (I-M-RC.3)

Current MOP: I-M-RC.3

```

```

I-M-CONTAMINATION-CAUSE.593 I-M-TEMPERATURE-OUT-OF-RANGE-CAUSE .599
I-M-MECHANICAL-SHOCK-CAUSE.605 I-M-THERMAL-SHOCK-CAUSE.611
I-M-ERRONEOUS-INPUT.4279 I-M-FAULTY-COMPONENT-CAUSE.3823)
* (quit)

```

```

Current MOP: COMP.1330 Input matches indices to: (I-M-RC.1 I-M-RC.2)
Current MOP: I-M-RC.1
Current MOP: I-M-RC.2
Current MOP: COMP.1329 Input matches indices to: (I-M-RC.1 I-M-RC.2)

Current MOP: I-M-RC.1
Current MOP: I-M-RC.2
Current MOP: COMP.4199 Input matches indices to: (I-M-TGU.1 I-M-RC.2)

Current MOP: I-M-TGU.1
Current MOP: I-M-TGU.2
Current MOP: I-M-TGU.3
Found: (COMP.2569 COMP.2499 COMP.1773 COMP.1321 COMP.4376 COMP.3574 COMP.3176
COMP.2505 COMP.1326 COMP.3573 COMP.3175 COMP.2504 COMP.1325 COMP.3572
COMP.3174 COMP.2503 COMP.1324 COMP.3571 COMP.3173 COMP.2502 COMP.1323
COMP.3570 COMP.3172 COMP.2501 COMP.1322 COMP.3575 COMP.3177 COMP.1779
COMP.3206 COMP.1327 COMP.2506 COMP.1781 COMP.1780 COMP.4378 COMP.4377 COMP.4202
COMP.4201 COMP.3988 COMP.3987 COMP.3719 COMP.3718 I-M-SDP.3 COMP.3577
COMP.3576 I-M-SDP.2 COMP.3179 I-M-SDP.1 COMP.3178 COMP.2792 COMP.2791
I-M-GW.3 COMP.2508 COMP.2507 I-M-GW.2 COMP.4375 I-M-GW.1 COMP.2134
COMP.1783 COMP.1782 I-M-RC.3 COMP.1330 COMP.1329 I-M-RC.1 COMP.1328
I-M-RC.2 I-M-TGU.1 I-M-TGU.2 H-CASE COMP.4199 I-M-TGU.3)

```

```

Values: ((COMP.3569 0) (COMP.2499 0) (COMP.1773 0) (COMP.1321 0) (COMP.4376 0)
(COMP.3574 0) (COMP.2505 0) (COMP.3176 0) (COMP.1326 0) (COMP.3573 0)
(COMP.3175 0) (COMP.2504 0) (COMP.1325 0) (COMP.3572 0) (COMP.3174 0)
(COMP.2503 0) (COMP.1324 0) (COMP.3571 0) (COMP.3173 0) (COMP.2502 0)
(COMP.1323 0) (COMP.3570 0) (COMP.3172 0) (COMP.2501 0) (COMP.1322 0)
(COMP.3575 0) (COMP.3177 0) (COMP.2506 0) (COMP.1325 0) (COMP.3576 0)
(COMP.3178 0) (COMP.2507 0) (COMP.1326 0) (COMP.3577 0) (COMP.3179 0)
(COMP.1781 0) (COMP.1780 0) (COMP.4378 0) (COMP.4377 0) (COMP.4202 0)
(COMP.4201 0) (COMP.3988 0) (COMP.3987 0) (COMP.3719 0) (COMP.3718 0)
(I-M-SDP.1) (COMP.3577 0) (COMP.3576 0) (COMP.3575 0) (COMP.3574 0)
(COMP.3179 0) (COMP.3178 0) (COMP.2940 0) (COMP.2940 0) (COMP.2792 0)
(COMP.2791 0) (COMP.2790 0) (COMP.2508 0) (COMP.2507 0) (COMP.2506 0)
(I-M-GW.2 533/588) (COMP.2135 0) (I-M-GW.1 2521/2940) (COMP.2134 0)
(COMP.1783 0) (COMP.1782 0) (I-M-RC.3 1) (COMP.1330 0) (COMP.1329 0)
(I-M-RC.1 2591/2940) (COMP.1328 0) (I-M-RC.2 533/588)
(I-M-TGU.1 2591/2940) (I-M-TGU.2 533/588) (M-CASE 0) (COMP.4199 0)
(I-M-TGU.3 1))

```

Picking out those that match 100%

Results of secondary elaboration:

- [1] PIECE-PART FAILURES
- [2] ERRONEOUS INPUT
- [3] CONTAMINATION
- [4] TEMPERATURE (HIGH OR LOW)
- [5] MECHANICAL SHOCK
- [6] THERMAL SHOCK

Elaborate using partial match? (y/n) n

```

(I-M-FAULTY-COMPONENT-CAUSE.2884 I-M-FAULTY-COMPONENT-CAUSE.1885
I-M-ERRONEOUS-INPUT.2594 I-M-FAULTY-COMPONENT-CAUSE.587

```

Appendix C : Code Listings

Contents of Code Listings

1. Instructions for Running FANSYS

Primary Instructions

- README

Help Files

- HelpFile
- myinspect.help

2. Initialization Code

Loading the Primary System

- fansys.lsp

Loading the X-Windows Q&A Interface

- cmumain.lsp
- cmutrace.lsp

Loading the Predefined Cases

- loademup

3. Domain Knowledge Representation

Entities

- entities.mops
- systems.mops

States

- states.mops

Actions

- actions.mops

Events

- events.mops

Procedures

- procedures.mops
- groups.mops
- causes.mops
- modes.mops
- detections.mops
- corrections.mops
- effects.mops

Cases

- cases.mops

Questions

- misc.mops

4. Memory Organization

Structure Definitions

- mem_structs.lsp

Memory Generalization and Retrieval Code

- mockmops6c.lsp

MOP Creation and Indexing Code

- memory.lsp

MOP Definition Macros

- mopdef_fns.lsp

5. The Parser

Parse and Activation Functions

- parser.lsp
- parser.mops

Lexical Pattern Management

- ip-fn.lsp
- hlpss.lsp
- hlpss.mops

Miscellaneous Functions

- failure.lsp
- failure.mops
- lisp_utils.lsp
- misc_utils.lsp
- predictions.lsp
- terminators.lsp

Generation

- generator.lsp

6. Domain Specific Parsing Knowledge

RC.1 Text

- rc.1.txt

Domain Lexical Knowledge

- entities.phrases
- events.phrases
- procedures.phrases
- groups.phrases
- causes.phrases
- modes.phrases
- detections.phrases
- corrections.phrases
- effects.phrases

RC.1 Processing Knowledge

- vocab.phrases

7. Question Answering

- Query Search and Retrieval
 - question.lsp
- Question Processing Knowledge
 - qa.phrases

8. User Interface

X-Windows Interface

- myinspect.lsp
- garnload.lsp
- interface.lsp
- title.lsp
- tclient.lsp
- tserver.lsp

Natural Language Question Shell

- qa.shell

9. FMEA Manual Cases

Ring Concentrator Cases

- cases.rc

Gateway Cases

- cases.gw

Standard Data Processor Cases

- cases.sdp

Time Generation Unit Cases

- cases.tgu

Fixed Multi-Purpose Applications Console Cases

- cases.fmpac

Mass Storage Device Cases

- cases.msd

English Case Descriptions

- cases.txt

6) FANSYS will continue to load. When it is ready, the system will instruct you to start the trace window. When this message appears, enter: (load "cmutrace") in the top window. This will instruct FANSYS to use the top window to display trace information. You will never enter further information into this window; it is an output window only for trace information.

7) FANSYS will load the Garnet and X-windows packages (CLX must already be installed). You will see the FANSYS logo in the upper right corner of the screen, and a panel of buttons in the lower right corner. These buttons were designed for a 1280x1024 size screen; if your screen is smaller, the buttons may appear jumbled or halfway off your screen. If so, you will either have to edit the Interface.lsp code to place the buttons where you desire, or you may move the buttons and Windows yourself manually after they appear.

After the files for FANSYS has loaded you may then execute FANSYS in either mode A (parse single case) or mode B (load hand-coded cases).

To run in mode A:

- 1) Examine the case to be parsed by typing : rc1. This will display the text for the first ring concentrator case.
- 2) Begin parsing the case by entering : (parse rc1). FANSYS will now parse the case. Use the inspector to browse memory (especially the m-case MOP) and to view the results.

The question shell permits a user to pose queries in natural language, rather than through the point and click interface. To run the shell, execute the following commands in the prompt Window:

```
(load "qa.phrases")
(load "qa.shell")
(start-shell)
```

You may now type in questions in English (provided they are in the format described in the Helpfile and in the accompanying tech report).

To run in mode B:

- 1) Enter : (load "loademup") This will load the 12 hand-coded cases into memory and will take anywhere from 10 to 60 minutes to complete, depending upon the speed of your system. It is often useful to save a core with these files pre-loaded so that FANSYS may be restarted at this point.
 - 2) Enter : (load "qa.phrases") to load the question phrases.
 - 3) Use the point-and-click buttons to ask FANSYS questions about these cases, or invoke the question shell.
 - 4) Click on "Quit" when finished.
- *****

Improving Performance:

The runtime speed and memory requirements of FANSYS will be dramatically improved if all of the .lsp files are compiled on your machine. Compilation should be performed AFTER the .lsp files have been initially loaded.

- *****
- 1) Open two X Windows, a top and bottom window.
 - 2) Start CMU Lisp in both windows.
 - 3) In the bottom window, type : (load "fansys"). This bottom window will be the main window for parsing and question answering. Loading will take several minutes. The fansys.lisp file contains the core knowledge base routines.
 - 4) After fansys.lsp has loaded, enter : (load "cmain"). This will load up the X-Windows and user interface routines.
 - 5) The inspector will pop up. Move the cursor to the inspector window and press 'd' to delete this window so loading may continue.
- This window appears to initialize the inspector routine.

alvarado@cs.ucdavis.edu
mock@cs.ucdavis.edu
braun@cs.ucdavis.edu

FANSYS - Failure Analysis System Prototype System v1.5

Help and Information

This prototype version of FANSYS is designed to answer questions about the Data Management System used in NASA's Space Station Freedom.

There are currently 12 cases represented. These include the Ring Concentrators, Gateway, Standard Data Processor, and the Time Generation Unit.

The Windows

Fansys communicates through two different windows. The lower window is where the main interaction takes place - prompting for user input and the output of information. The upper window contains trace information which shows what the system is doing on a technical level.

The Interface

Two different interfaces are currently under development: an X11 User Interface, and a text interface.

The X11 Interface is based on buttons.

There are 4 major button types:

- 1) Enter a search query. By clicking on this button, the user will be presented with a variety of menus prompting for the requested information and given information with which to perform a search query.

- 2) Inspect memory. This buttons brings up the graphical inspector, which allows a user to view the memory hierarchy.
Press '?' for a list of commands.
- 3) Help. Shows this help file.
- 4) Quit. Ends the session.

The Text Interface

Questions may also be entered in textual format. All questions should be of the following form:
what is the (requested-item) [and the (requested-item)] [for the (given-item)] [when the (item) is (specifier)] [and the (item) is (specifier)] ?

Note that the question mark must be separated from the last word by a blank space.

For example,

What is the failure mode for the gateway ?

What is the failure cause for the standard data processor when the failure mode is loss of output - failure to start ?

What is the failure cause and the failure detection procedure for the ring concentrator ?

The requested-items may be any of:

failed component
failure mode
failure cause
detection procedure
correction procedure
failure effect

The given-item may be any of:

ring concentrator
gateway
standard data processor
time generation unit
mass storage device

The item and corresponding specifier pairs may be any of:

item:

specifier:

failed component
ring concentrator
gateway
standard data processor
time generation unit
mass storage device

failure mode
loss of output - failure to start
loss of output - failure during operation
operational - erroneous output

failure cause
thermal shock
piece-part failures
temperature *left-paren* high or low *right-paren*
mechanical shock
erroneous input

detection procedure
next node detection
heartbeat messages
error correction code detection
power-on self test

correction procedure
check connections or replace
automatic backup from lookup-table
network self-reconfiguration
replace with spare
switch or bypass via backup net

Press 'D' when done reading to destroy this window.

The component objects of the window's object will become highlighted (surrounded by a box) as the mouse passes over them. In an inspector window, keystrokes and mouse clicks are interpreted as follows:

- Left** When the mouse is over a component object, clicking Left will inspect that object in a new inspector window. If the object clicked upon is a MOP, that MOP will be expanded and displayed.
- Middle** This is identical to clicking on Left, except the new object is displayed in the current inspector window rather than in a new window. The "up" command (below) can be used to return to the current object.
- Right** Clicking Right will close all windows and exit the inspector.
- d, D** Typing "d" or "D" inside an inspector window will delete that window, and exit the inspector if there are no more inspector windows.
- h, H, ?** Typing "h", "H", or "?" inside an inspector window will create a window the helpful instructions you are reading now.
- m, M** Typing "m" or "M" inside an inspector window will allow one to modify a component of an object. The mouse cursor will change from an arrow to an arrow with an "M" beside it, indicating that one should select the component to be modified. Clicking any mouse button when the mouse is over a component will select that component as a destination for modification.
- If one has typed "m", the source object will also be selected by the mouse, with the mouse cursor changed to an arrow with an "S" beside it. The object will replace the destination component.
- If one has typed "m", the source object will be prompted for on the "Query-IO" stream.
- When choosing the destination or source with the mouse, one may type "q" or "Q" to abort the modify operation.
- q, Q** Typing "q" or "Q" will quit the inspector, deleting all new inspector windows.
- p, P** Typing "p" or "P" will proceed from the inspector, leaving all inspector windows intact.
- r, R** Typing "r" or "R" will recompute the display for the object in the window. This is used to maintain a consistent display for an object that may have changed since the display was computed.
- u, U** Typing "u" or "U" takes one back up the chain of investigation, to the object for which this object was displayed as a component. This only

works for displays generated by modifying a previously current display; this does not work for a display generated as a new inspector window.

When you are done reading Help, Press "d" to delete this window.

```

*****  

; FANSYS: A Computer Model of Text Comprehension and Question  

; Answering for Failure Analysis  

;  

; File: fansys.lisp  

;  

; Developed by: Sergio J. Alvarado  

; Ronald K. Braun  

; Kenrick J. Mock  

;  

; Artificial Intelligence Laboratory  

; Computer Science Department  

; University of California  

; Davis, CA 95616  

;  

; Funds for the support of this study have been allocated by the  

; NASA-Ames Research Center, Moffett Field, California, under  

; Interchange No. NCA2-721.  

;  

; Also global variables defined here  

;  

(format t "-FANSYS -- Version 1.6")  

; some debugging statements exist -- set *debug* to t to turn them on  

; Also global variables defined here  

(defvar *debug* nil)  

(defvar *generalize* nil)  

(defvar *verbose* nil)  

(defvar *mops* nil)  

(defvar *mode* nil)  

(defvar *exploded* nil)  

(defvar *generate* nil)  

(defvar *templist* nil)  

;  

; utilities  

(load "parser/lisp_utils")  

(load "parser/mem_structs")  

;  

; mop and AM/PM memory structures  

(load "parser/mem_structs")  

;  

; memory access functions and initialize memory  

;  

; hash table for exploded mops  

(setf *Exploded* (make-hash-table))  

(load "memory/mockmops6c")  

(load "memory/question")  

(clear-memory)  

;  

; load "parser/parser.mops"  

(load "parser/hlips.mops")  


```

```
*****  
/*  
 * FANSYS: A Computer Model of Text Comprehension and Question  
 * Answering for Failure Analysis  
 */  
  
File: cmumain.lsp  
*****  
/* Developed by: Sergio J. Alvarado  
 * Ronald K. Braun  
 * Kenrick J. Mock  
 */  
/* Artificial Intelligence Laboratory  
 * Computer Science Department  
 * University of California  
 * Davis, CA 95616  
 */  
/* Funds for the support of this study have been allocated by the  
 * NASA-Ames Research Center, Moffett Field, California, under  
 * Interchange No. NC42-721.  
 */  
*****  
/* Load the modified inspector  
(load "memory/myinspect")  
/* Open up a window to initialize Inspector so we can do the help window  
(format t "~&Should remove this inspector line with recompiled core~&")  
(inspect (sh 'm-root))  
/* Load server file for trace window  
(load "sockets/tserver")  
/* Load Garnet Code  
/* This will take a while.  
(load "/net/epoch/usr/aceugrad/mock/garnet/garnet-loader")  
/* Load various garnet windows  
(load "garnet/interface")  
/* Load title stuff  
(load "garnet/title")
```

```
*****  
** FANSYS: A Computer Model of Text Comprehension and Question  
** Answering for Failure Analysis  
**  
** File: cmtrace.lsp  
**  
** Developed by: Sergio J. Alvarado  
** Ronald K. Braun  
** Kenrick J. Mock  
**  
** Artificial Intelligence Laboratory  
** Computer Science Department  
** University of California  
** Davis, CA 95616  
**  
** Funds for the support of this study have been allocated by the  
** NASA-Ames Research Center, Moffett Field, California, under  
** Interchange No. NCA2-721.  
**  
** Load the client trace program. Make sure the server is already  
** running first.  
**  
(load "sockets/tclient")
```

```
*****  
/*  
 * FANSYS: A Computer Model of Text Comprehension and Question  
 * Answering for Failure Analysis  
 */  
*****  
/* File: loademup  
 */  
/* Developed by: Sergio J. Alvarado  
 * Ronald K. Braun  
 * Kenrick J. Mock  
 */  
/* Artificial Intelligence Laboratory  
 * Computer Science Department  
 * University of California  
 * Davis, CA 95616  
 */  
/* Funds for the support of this study have been allocated by the  
 * NASA-Ames Research Center, Moffett Field, California, under  
 * Interchange No. NCA2-721.  
 */  
*****  
/*  
 (load "cases/cases.rc")  
 (load "cases/cases.qw")  
 (load "cases/cases.sdp")  
 (load "cases/cases.tgu")  
 */
```

```

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: entities.mops
***** Developed by: Sergio J. Alvarado
***** Ronald K. Braun
***** Kentick J. Mock
***** Artificial Intelligence Laboratory
***** Computer Science Department
***** University of California
***** Davis, CA 95616
***** Funds for the support of this study have been allocated by the
***** NASA-Ames Research Center, Moffett Field, California, under
***** Interchange No. NCA2-721.
; Domain knowledge representation for the DMS of the Space Station Freedom:
; Abstract representation of DMS entities.

; m-DMS-entity is defined in systems.mops

(defmop m-message (m-DMS-entity))
(defmop m-token (m-message))
(defmop m-packet (m-message))
(defmop m-heartbeat-message (m-message))
(defmop m-status-message (m-message))
(defmop m-time-out-message (m-message))
(defmop m-erroneous-message (m-message))
(defmop m-null-message (m-message))
(defmop m-generic-message (m-message))

(defmop m-network (m-system))
(defmop m-JEM (m-network))
(defmop m-ESA (m-network))
(defmop m-OMS (m-network))
(defmop m-payload-network (m-network))

(defmop m-core-network-backup (m-network))
(defmop m-ORU (m-hardware-component))
(defmop m-transmitter (m-hardware-component))
(defmop m-connectors (m-hardware-component))
(defmop m-power-system (m-hardware-component))
(defmop m-RC (m-ORU)
  (software m-SM)
  (hardware m-transmitter))

(defmop m-generic-rc (m-rc))
(defmop m-next-rc (m-rc))
(defmop m-backup-rc (m-rc))
(defmop m-GW (m-ORU)
  (software m-SM)
  (software m-ECC))
(defmop m-backup-gw (m-gw))
(defmop m-SDP (m-ORU)
  (software m-SM)
  (hardware m-BCU)
  (hardware m-BIA)
  (hardware m-local-bus))

(defmop m-generic-SDP (m-SDP))
(defmop m-backup-SDP (m-SDP))
(defmop m-TCU (m-ORU)
  (hardware m-TDB))
(defmop m-generic-TGU (m-TGU))
(defmop m-backup-TGU (m-TGU))
(defmop m-FMPAC (m-ORU)
  (software m-SM)
  (software m-ECC))
(defmop m-generic-FMPAC (m-FMPAC))
(defmop m-backup-FMPAC (m-FMPAC))
(defmop m-MSD (m-ORU))
(defmop m-generic-MSD (m-MSD))
(defmop m-backup-MSD (m-MSD))
(defmop m-CMPAC (m-ORU)
  (software m-SM)
  (software m-BCC)
  (hardware m-cupola))
(defmop m-generic-CMPAC (m-CMPAC))
(defmop m-backup-CMPAC (m-CMPAC))
(defmop m-DMS (m-system)
  (hardware m-RC)
  (software m-OMS))

(defmop m-mission-component (m-system))
(defmop m-crew (m-mission-component))
(defmop m-mission-support (m-mission-component))
(defmop m-systems (m-mission-component))
(defmop m-interfaces (m-mission-component))

(defmop m-TDB (m-hardware-component))
(defmop m-lookup-table (m-software-component))
(defmop m-JEM (m-software-component))
(defmop m-OMA (m-software-component))
(defmop m-PMAD (m-software-component))
(defmop m-ECC (m-software-component))
(defmop m-POST (m-software-component))
(defmop m-local-bus (m-hardware-component))
(defmop m-RC (m-hardware-component))

```

```
*****
;*
;* FANSYS: A Computer Model of Text Comprehension and Question
;* Answering for Failure Analysis
;*
;* File: systems.mops
;*
;* Developed by: Sergio J. Alvarado
;*                 Ronald K. Braun
;*                 Kenick J. Mock
;*
;* Artificial Intelligence Laboratory
;* Computer Science Department
;* University of California
;* Davis, CA 95616
;*
;* Funds for the support of this study have been allocated by the
;* NASA-Ames Research Center, Moffett Field, California, under
;* Interchange No. NCA2-721.
;*
;* ****
;*
;* Domain knowledge representation for the DMS of the Space Station Freedom:
;* Abstract representation of a system.
;
(defmop m-DMS-entity (m-root))
;
(defmop m-system (m-DMS-entity m-question-constraint))
;
(defmop m-software-component (m-system)
  (software m-software-component))
;
(defmop m-hardware-component (m-system)
  (hardware m-hardware-component)
  (software m-software-component))
```

```

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: states.mops

Developed by: Sergio J. Alvarado
               Ronald K. Braun
               Kenrick J. Mock

Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616

Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

; Domain knowledge representation for the DMS of the Space Station Freedom:
; Abstract representation of states.

(defmop m-state (m-root))

(defmop m-power-state (m-state)
  (object m-root))

(defmop m-power-on (m-power-state))
(defmop m-power-off (m-power-state))

(defmop m-diagnostic-result-state (m-state)
  (diagnostic m-root)
  (object m-root))

(defmop m-diagnostic-pass (m-diagnostic-result-state))
(defmop m-diagnostic-fail (m-diagnostic-result-state))

(defmop m-time-for-object-state (m-state)
  (object m-root))

(defmop m-time-elapsed-for-object (m-time-for-object-state))
(defmop m-time-exceeded-for-object (m-time-for-object-state))
(defmop m-time-equivalence (m-time-for-object-state)
  (object1 m-root)
  (object2 m-root))

(defmop m-operational-state (m-state)
  (object m-system))

(defmop m-operational (m-operational-state))
(defmop m-non-operational (m-operational-state))
(defmop m-unknown-operational (m-operational-state))

(defmop m-transmission-state (m-state)
  (sender m-root)
  (recipient m-root)
  (object m-message))

(defmop m-ready-to-transmit (m-transmission-state))
(defmop m-ready-to-receive (m-transmission-state))

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: states.mops

Developed by: Sergio J. Alvarado
               Ronald K. Braun
               Kenrick J. Mock

Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616

Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

; Domain knowledge representation for the DMS of the Space Station Freedom:
; Abstract representation of states.

(defmop m-state (m-root))

(defmop m-power-state (m-state)
  (object m-root))

(defmop m-power-on (m-power-state))
(defmop m-power-off (m-power-state))

(defmop m-diagnostic-result-state (m-state)
  (diagnostic m-root)
  (object m-root))

(defmop m-diagnostic-pass (m-diagnostic-result-state))
(defmop m-diagnostic-fail (m-diagnostic-result-state))

(defmop m-time-for-object-state (m-state)
  (object m-root))

(defmop m-time-elapsed-for-object (m-time-for-object-state))
(defmop m-time-exceeded-for-object (m-time-for-object-state))
(defmop m-time-equivalence (m-time-for-object-state)
  (object1 m-root)
  (object2 m-root))

(defmop m-operational-state (m-state)
  (object m-system))

(defmop m-operational (m-operational-state))
(defmop m-non-operational (m-operational-state))
(defmop m-unknown-operational (m-operational-state))

(defmop m-transmission-state (m-state)
  (sender m-root)
  (recipient m-root)
  (object m-message))

(defmop m-ready-to-transmit (m-transmission-state))
(defmop m-ready-to-receive (m-transmission-state))

```

```

*****
***** FANSYS: A Computer Model of Text Comprehension and Question
***** Answering for Failure Analysis
*****
** File: actions.mops
**
** Developed by: Sergio J. Alvarado
**                 Ronald K. Braun
**                 Kenrick J. Mock
**
** Artificial Intelligence Laboratory
** Computer Science Department
** University of California
** Davis, CA 95616
**
** Funds for the support of this study have been allocated by the
** NASA-Ames Research Center, Moffett Field, California, under
** Interchange No. NCA2-721.
**
***** Domain knowledge representation for the DMS of the Space Station Freedom:
; Abstract representation of actions.
*****
(defmop m-action (m-root)
  (actor m-root)
  (object m-root))

(defmop m-transmission-action (m-action)
  (sender m-root)
  (recipient m-root))

(defmop m-message-transaction (m-transmission-action))
(defmop m-prepare-to-receive (m-transmission-action))
(defmop m-wait-for-message (m-transmission-action))
(defmop m-time-out (m-transmission-action))

(defmop m-power-action (m-action))

(defmop m-power-up (m-power-action))
(defmop m-power-down (m-power-action))

(defmop m-change-location (m-action)
  (to m-root)
  (from m-root))

(defmop m-diagnostic-action (m-action))

(defmop m-do-POST (m-diagnostic-action))
(defmop m-do-ECC (m-diagnostic-action))

(defmop m-connection-action (m-action)
  (object1 m-root)
  (object2 m-root))

(defmop m-physical-connect (m-connection-action))
(defmop m-physical-disconnect (m-connection-action))

(defmop m-logical-connect (m-connection-action))
(defmop m-logical-disconnect (m-connection-action))

(defmop m-look-up-replacement (m-action)

```

```

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: events.mops
Developed by: Sergio J. Alvarado
               Ronald K. Braun
               Kenrick J. Mock
Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616
Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

; Domain knowledge representation for the DMS of the Space Station Freedom:
; Abstract representation of events.

(defmop m-event (m-event-step)
  (precond m-state)
  (action m-action)
  (result m-state))

(defmop m-null-event (m-event)
  (sender m-root)
  (recipient m-root)
  (action m-null-action))

(defmop m-transmission-event (m-event)
  (sender m-root)
  (recipient m-root)
  (message m-message))

(defmop m-send-message-event (m-transmission-event)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (message m-message MESSAGE)
  (precond m-ready-to-transmit)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (object m-message MESSAGE))
  (result m-message-transmitted)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (object m-message MESSAGE))
  (m-transmission-event)

(defmop m-receive-message-event (m-transmission-event)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (message m-message MESSAGE)
  (precond m-ready-to-receive)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (object m-message MESSAGE))
  (m-transmission-event)

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: events.mops
Developed by: Sergio J. Alvarado
               Ronald K. Braun
               Kenrick J. Mock
Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616
Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

; Domain knowledge representation for the DMS of the Space Station Freedom:
; Abstract representation of events.

(defmop m-event (m-event-step)
  (precond m-state)
  (action m-action)
  (result m-state))

(defmop m-null-event (m-event)
  (sender m-root)
  (recipient m-root)
  (action m-null-action))

(defmop m-transmission-event (m-event)
  (sender m-root)
  (recipient m-root RECEIVER)
  (message m-message MESSAGE))
  (m-transmission-event)

(defmop m-wait-to-receive-event (m-transmission-event)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (message m-message MESSAGE)
  (precond m-ready-to-receive)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (object m-message MESSAGE))
  (action m-wait-for-message)
  (factor m-root RECEIVER)
  (object m-message MESSAGE))
  (m-transmission-event)

(defmop m-prepare-to-receive-event (m-transmission-event)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (message m-message MESSAGE)
  (action m-prepare-to-receive)
  (factor m-root RECEIVER)
  (object m-message MESSAGE))
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (object m-message MESSAGE))
  (m-transmission-event)

(defmop m-wait-to-transmit-event (m-transmission-event)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (message m-message MESSAGE)
  (precond m-ready-to-transmit)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (object m-message MESSAGE))
  (action m-wait-for-message)
  (actor m-root SENDER)
  (recipient m-root RECEIVER)
  (sender m-root SENDER)
  (object m-message MESSAGE))
  (m-transmission-event)

(defmop m-prepare-to-transmit-event (m-transmission-event)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (message m-message MESSAGE))
  (precond m-ready-to-transmit)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (object m-message MESSAGE))
  (result m-message-transmitted)
  (sender m-root SENDER)
  (recipient m-root RECEIVER)
  (object m-message MESSAGE))
  (m-transmission-event)

```

```

(result m-time-lapsed-for-object
  (object m-root SENDER))
(defmop m-time-out-event (m-transmission-event)
  (sender m-root SENDER)
  (recipient m-system RECEIVER)
  (message m-message MESSAGE)
  (precond m-ready-to-receive
    (sender m-root SENDER)
    (recipient m-root RECEIVER)
    (object m-message MESSAGE)))
  (precond m-time-equivalence
    (object1 m-time-elapses-for-object
      (object m-root RECEIVER)
      (object2 m-root RECEIVER))
    (object2 m-time-exceeds-for-object
      (object m-root RECEIVER)))
  (action m-time-out
    (actor m-root RECEIVER)
    (sender m-root SENDER)
    (recipient m-root RECEIVER)
    (object m-message MESSAGE)))
  (result m-transmission-timed-out
    (sender m-root SENDER)
    (recipient m-root RECEIVER)
    (object m-message MESSAGE)))
(defmop m-power-event (m-event)
  (actor m-root)
  (object m-root))
(defmop m-power-up-event (m-power-event)
  (actor m-root)
  (object m-root)
  (precond m-power-on
    (object m-root))
  (action m-power-up
    (actor m-root)
    (object m-root)
    (object m-power-on)
    (object m-root)))
  (result m-power-on
    (object m-root)))
(defmop m-power-down-event (m-power-event)
  (actor m-root ACTOR)
  (object m-root OBJECT)
  (precond m-power-on
    (object m-root OBJECT))
  (action m-power-down
    (actor m-root ACTOR)
    (object m-root OBJECT))
  (result m-power-off
    (object m-root OBJECT)))
(defmop m-diagnostic-event (m-event)
  (actor m-root)
  (object m-root))
(defmop m-do-POST-event (m-diagnostic-event)
  (actor m-root ACTOR)
  (object m-root OBJECT)
  (action m-do-POST
    (actor m-root ACTOR)
    (object m-root OBJECT)))
  (defmop m-ddo-ECC-event (m-diagnostic-event)
    (actor m-root ACTOR)
    (object m-root OBJECT)
    (action m-do-ECC
      (actor m-root ACTOR)
      (object m-root OBJECT))))
```

```

(actor m-root ACTOR)
  (object1 m-root OBJECT1)
  (object2 m-root OBJECT2)
  (result m-physically-connected)
  (object1 m-root OBJECT1)
  (object2 m-root OBJECT2))

(defmop m-look-up-replacement-event (m-event)
  (actor m-root ACTOR)
  (object m-root BACKUP-NODE)
  (backed-up m-root FAILED-NODE)
  (from m-root SOURCE)
  (action m-look-up-replacement)
  (actor m-root ACTOR)
  (object m-root BACKUP-NODE)
  (backed-up m-root FAILED-NODE)
  (from m-root SOURCE)
  (to m-root ACTOR))

(defmop m-verify-event (m-event)
  (actor m-system)
  (object m-system))

(defmop m-verify-power-event (m-verify-event)
  (actor m-system ACTOR)
  (object m-power-system OBJECT)
  (action m-verify-power)
  (actor m-root ACTOR)
  (object m-power-system OBJECT))

(defmop m-verify-connectors-event (m-verify-event)
  (actor m-system ACTOR)
  (object m-connectors OBJECT)
  (action m-verify-connectors)
  (actor m-root ACTOR)
  (object m-connectors OBJECT))

```

```

*****
;* FANSYS: A Computer Model of Text Comprehension and Question
;* Answering for Failure Analysis
;*
;* File: procedures.mops
;*
;* Developed by: Sergio J. Alvarado
;* Ronald K. Braun
;* Kenrick J. Mock
;*
;* Artificial Intelligence Laboratory
;* Computer Science Department
;* University of California
;* Davis, CA 95616
;*
;* Funds for the support of this study have been allocated by the
;* NASA-Ames Research Center, Moffett Field, California, under
;* Interchange No. NCA2-721.
;*
;*
;* Domain knowledge representation for the DMS of the Space Station Freedom:
;* Abstract representation of sequences of events and procedures.
;*
;* (defmop m-sequence-of-events (m-root m-question-constraint)
;*   (precond m-state)
;*   (sequence-of-steps m-group)
;*   (result m-state))
;*
;* (defmop m-procedure (m-sequence-of-events m-event-step)
;*   (sequence-of-steps m-group))
;*
;* (defmop m-detection-proc (m-procedure))
;* (defmop m-verification-proc (m-procedure))
;* (defmop m-notification-proc (m-procedure))
;* (defmop m-correction-proc (m-procedure))
;*
;* (defmop m-null-and-group (m-and-group)
;*   (1 m-null-event))
;*
;* (defmop 1-m-null-procedure (m-procedure) instance
;*   (sequence-of-steps m-null-and-group instance
;*     (1 m-null-event instance
;*       (action 1-m-null-action))))

```

```
*****  
/*  
 * FANSYS: A Computer Model of Text Comprehension and Question  
 * Answering for Failure Analysis  
 */  
  
/* File: groups.mops  
 */  
  
/* Developed by: Sergio J. Alvarado  
 * Ronald K. Braun  
 * Kenrick J. Mock  
 */  
  
/* Artificial Intelligence Laboratory  
 * Computer Science Department  
 * University of California  
 * Davis, CA 95616  
 */  
  
/* Funds for the support of this study have been allocated by the  
 * NASA-Ames Research Center, Moffett Field, California, under  
 * Interchange No. NCA2-721.  
 */  
  
*****  
;  
; Domain knowledge representation for the DMS of the Space Station Freedom:  
; Abstract representation of groups.  
  
(defmop m-event-step (m-root))  
(defmop m-group (m-event-step))  
(defmop m-and-group (m-group)  
  (l m-event-step))  
(defmop m-or-group (m-group)  
  (l m-event-step))
```

```

***** File: causes.mops *****

; FANSYS: A Computer Model of Text Comprehension and Question
; Answering for Failure Analysis
;
; File: causes.mops
;
; Developed by: Sergio J. Alvarado
;                 Ronald K. Braun
;                 Kenrick J. Mock
;
; Artificial Intelligence Laboratory
; Computer Science Department
; University of California
; Davis, CA 95616
;
; Funds for the support of this study have been allocated by the
; NASA-Ames Research Center, Moffett Field, California, under
; Interchange No. NCA2-721.
;
; Domain knowledge representation for the DMS of the Space Station Freedom:
; Abstract representation of failure causes.

(defmop m-failure-cause (m-failure-cause)
  (failed-component m-root FAILED-COMPONENT))

(defmop m-faulty-component-cause (m-failure-cause)
  (failed-component m-root FAILED-COMPONENT)
  (precond m-operational
    (object m-root FAILED-COMPONENT))
  (result m-non-operational
    (object m-root FAILED-COMPONENT)))

(defmop m-contamination-cause (m-failure-cause)
  (failed-component m-root FAILED-COMPONENT)
  (precond m-uncontaminated
    (object m-root FAILED-COMPONENT))
  (result m-contaminated
    (object m-root FAILED-COMPONENT)))

(defmop m-temperature-out-of-range-cause (m-failure-cause)
  (failed-component m-root FAILED-COMPONENT)
  (precond m-temperature-tolerable
    (object m-root FAILED-COMPONENT))
  (result m-temperature-intolerable
    (object m-root FAILED-COMPONENT)))

(defmop m-mechanical-shock-cause (m-failure-cause)
  (failed-component m-root FAILED-COMPONENT)
  (precond m-structurally-intact
    (object m-root FAILED-COMPONENT))
  (result m-structurally-broken
    (object m-root FAILED-COMPONENT)))

(defmop m-thermal-shock-cause (m-failure-cause)
  (failed-component m-root FAILED-COMPONENT)
  (precond m-structurally-intact
    (object m-root FAILED-COMPONENT))
  (result m-structurally-broken
    (object m-root FAILED-COMPONENT)))

```

```

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: modes.mops

Developed by: Sergio J. Alvarado
               Ronald K. Braun
               Kenrick J. Mock

Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616

Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

; Domain knowledge representation for the DHS of the Space Station Freedom:
; Abstract representation of the failure modes.

(defmop m-failure-mode (m-or-u)
  (failed-component m-or-u))

(defmop m-startup-no-output (m-failure-mode)
  (failed-component m-root FAILED-COMPONENT)
  (detection-node m-root DETECTION-COMPONENT)
  (message m-message MESSAGE)
  (precond m-ready-to-receive
    (sender m-root FAILED-COMPONENT)
    (recipient m-root DETECTION-COMPONENT)
    (object m-message MESSAGE)))
  (sequence-of-steps m-and-group
    (1 m-power-up-event
      (object m-root FAILED-COMPONENT))
    (2 m-wait-to-receive-event
      (sender m-root FAILED-COMPONENT)
      (recipient m-root DETECTION-COMPONENT)
      (message m-message MESSAGE)))
  (3 m-time-out-event
    (sender m-root FAILED-COMPONENT)
    (recipient m-root DETECTION-COMPONENT)
    (message m-message MESSAGE)))

(defmop m-operational-no-output (m-failure-mode)
  (failed-component m-root FAILED-COMPONENT)
  (detection-node m-root DETECTION-COMPONENT)
  (message m-message MESSAGE)
  (precond m-operational
    (object m-root FAILED-COMPONENT)
    (precond m-ready-to-receive
      (sender m-root FAILED-COMPONENT)
      (recipient m-root DETECTION-COMPONENT)
      (object m-message MESSAGE)))
  (sequence-of-steps m-and-group
    (1 m-wait-to-receive-event
      (sender m-root FAILED-COMPONENT)
      (recipient m-root DETECTION-COMPONENT)
      (message m-message MESSAGE)))))

*****
```



```

(message m-erroneous-message instance MESSAGE)
(2 m-do-ECC-event
  (actor m-root DETECTION-NODE)
  (object m-erroneous-message MESSAGE)
  (result m-diagnostic-fail
    (diagnostic m-ECC instance DIAGNOSTIC)
    (object m-root SUSPECT-NODE))))))
(result m-non-operational
  (object m-ORU SUSPECT-NODE)))
(3 m-drop m-heartbeat-detection (m-failure-detection)
  (failed-component m-root SUSPECT-NODE)
  (detection-component m-SM DETECTION-NODE)
  (precond m-physically-connected
    (object m-root DETECTION-NODE)
    (object2 m-network)
    (object2 m-network)))
  (precond m- logically-connected
    (object1 m-root DETECTION-NODE)
    (object2 m-network)))
  (precond m-power-on
    (object m-root DETECTION-NODE))
  (sequence-of-steps m-detection-and-group
    (1 m-detection-proc
      (sequence-of-steps m-and-group
        (1 m-send-message-event
          (sender m-root DETECTION-NODE)
          (recipient m-root SUSPECT-NODE)
          (message m-heartbeat-message instance MESSAGE)))
        (2 m-prepare-to-receive-event
          (sender m-root SUSPECT-NODE)
          (recipient m-root DETECTION-NODE)
          (message m-status-message instance STATUS)))
        (3 m-wait-to-receive-event
          (sender m-root SUSPECT-NODE)
          (recipient m-root DETECTION-NODE)
          (message m-status-message STATUS)))
        (4 m-time-out-event
          (sender m-root SUSPECT-NODE)
          (recipient m-root DETECTION-NODE)
          (message m-status-message STATUS)))))))
(result m-non-operational
  (object m-ORU SUSPECT-NODE)))

```



```

(configuration m-DMS SYSTEM)
  (object2 m-root BACKUP)
  (action m-logical-connect)
  (actor m-root CORRECTION)
  (object1 m-root BACKCOMP)
  (object2 m-root BACKUP)
  (result m-logical-connected)
  (object1 m-root BACKCOMP)
  (object2 m-root BACKUP)))
  (3 m-logical-connect-event
    (actor m-sm CORRECTION)
    (object1 m-network BACKUP)
    (object2 m-DMS SYSTEM)
    (precond m-logically-disconnected)
    (object1 m-root BACKUP)
    (object2 m-root SYSTEM))
    (object2 m-root BACKUP)
    (action m-logical-connect
      (actor m-root CORRECTION)
      (object1 m-root BACKUP)
      (object2 m-root SYSTEM))
      (result m-logically-connected)
      (object1 m-root BACKUP)
      (object2 m-root SYSTEM)))))

(defmop m-bypass-node (m-reconfiguration)
  (correction-component m-sm CORRECTION)
  (failed-component m-root FAILED)
  (configuration m-network NETWORK)
  (sequence-of-steps m-correction-and-group
    (1 m-correction-proc
      (sequence-of-steps m-and-group
        (1 m-logical-disconnect-event
          (actor m-sm CORRECTION)
          (object1 m-root FAILED)
          (object2 m-network NETWORK)
          (precond m-logically-connected)
          (object1 m-root FAILED)
          (object2 m-root NETWORK)
          (action m-logical-disconnect
            (actor m-root CORRECTION)
            (object1 m-root FAILED)
            (object2 m-root NETWORK))
          (result m-logically-disconnected)
          (object1 m-root FAILED)
          (object2 m-root NETWORK)))
        (result m-operational
          (object m-network NETWORK))))))

(defmop m-bypass (m-reconfiguration)
  (correction-component m-sm CORRECTION)
  (failed-component m-root FAILED)
  (configuration m-network NETWORK)
  (sequence-of-steps m-correction-and-group
    (1 m-correction-proc
      (sequence-of-steps m-or-group
        (1 m-logical-switch
          (correction-component m-sm CORRECTION)
          (failed-component m-network NETWORK)
          (backup-component m-root BACKCOMP)
          (precond m-logically-connected)
          (object1 m-root FAILED)
          (object2 m-root NETWORK)
          (sequence-of-steps m-correction-and-group
            (1 m-correction-proc
              (sequence-of-steps m-and-group
                (1 m-physical-disconnect-event
                  (actor m-crew CORRECTION)
                  (object1 m-root FAILED)
                  (object2 m-network NETWORK)))
              (4 m-logical-switch
                (correction-component m-system CORRECTION)
                (failed-component m-system FAILED)
                (backup-component m-system BACKUP)
                (configuration m-network NETWORK))))))))
    (defmop m-replace-with-spare (m-failure-correction)
      (correction-component m-crew CORRECTION)
      (failed-component m-root FAILED)
      (configuration m-network NETWORK)
      (sequence-of-steps m-correction-and-group
        (1 m-correction-proc
          (sequence-of-steps m-and-group
            (1 m-physical-connect-event
              (actor m-crew CORRECTION)
              (object1 m-root FAILED)
              (object2 m-network NETWORK)))
        (4 m-physical-connect-event
          (correction-component m-network NETWORK)))))))

(defmop m-replace (m-failure-correction)
  (correction-component m-crew CORRECTION)
  (failed-component m-root FAILED)
  (configuration m-network NETWORK)
  (sequence-of-steps m-correction-and-group
    (1 m-correction-proc
      (sequence-of-steps m-and-group
        (1 m-physical-disconnect-event
          (actor m-crew CORRECTION)
          (object1 m-root FAILED)
          (object2 m-network NETWORK)))))))

```

```

(actor m-crew CORRECTION)
  (object1 m-root BACKUP)
  (object2 m-network NETWORK)
  (3 m-logical-disconnect-event
    (actor m-root CORRECTION)
    (object1 m-root FAILED)
    (object2 m-root NETWORK)
    (precond m-logically-connected
      (object1 m-root FAILED)
      (object2 m-root FAILED))
    (object2 m-root NETWORK)
    (action m-logical-disconnect
      (actor m-root CORRECTION)
      (object1 m-root FAILED)
      (object2 m-root FAILED))
    (object2 m-root NETWORK))
  (4 m-logical-connect-event
    (actor m-root CORRECTION)
    (object1 m-root BACKUP)
    (object2 m-root BACKUP)
    (object2 m-root NETWORK)
    (precond m-logically-disconnected
      (object1 m-root BACKUP)
      (object2 m-root NETWORK))
    (action m-logical-connect
      (actor m-root CORRECTION)
      (object1 m-root BACKUP)
      (object2 m-root BACKUP))
    (object2 m-root NETWORK)
    (result m-logically-connected
      (object1 m-root BACKUP)
      (object2 m-root BACKUP)))
    (object2 m-root NETWORK)))
  (result m-logically-connected
    (object1 m-root BACKUP)
    (object2 m-system NETWORK))
  (result m-logically-disconnected
    (object1 m-root FAILED)
    (object2 m-system NETWORK))
  (precond m-non-operational
    (object m-root FAILED))
  (object m-root FAILED)
  (result m-operational
    (object m-root BACKUP)))
  (object m-root NETWORK))

(defmop m-non-operational-replacement (m-failure-correction)
  (correction-component m-crew CORRECTION)
  (failed-component m-root FAILED)
  (configuration m-network NETWORK)
  (backup-component m-root BACKUP)
  (precond m-non-operational
    (object m-root FAILED)))
  (sequence-of-steps m-correction-and-group
    (1 m-correction-proc
      (sequence-of-steps m-and-group
        (1 m-and-group
          (1 m-verify-power-event
            (actor m-crew CORRECTION)
            (object m-power-system instance))
          (2 m-verify-connectors-event
            (actor m-crew CORRECTION)
            (object m-connectors instance)))
          (3 m-replace-with-spare
            (correction-component m-crew CORRECTION)
            (failed-component m-root FAILED)
            (configuration m-network NETWORK)
            (backup-component m-root BACKUP)))))))

```

```
*****  
;* FANSYS: A Computer Model of Text Comprehension and Question  
;* Answering for Failure Analysis  
;*  
;* File: effects.mops  
;  
;* Developed by: Sergio J. Alvarado  
;* Ronald K. Braun  
;* Kenrick J. Mock  
;  
;* Artificial Intelligence Laboratory  
;* Computer Science Department  
;* University of California  
;* Davis, CA 95616  
;  
;* Funds for the support of this study have been allocated by the  
;* NASA-Ames Research Center, Moffett Field, California, under  
;* Interchange No. NCA2-721.  
;  
*****  
; Domain knowledge representation for the DMS of the Space Station Freedom:  
; Abstract representation of an effect.  
;  
(defmop m-failure-effect (m-procedure)  
  (sequence-of-steps m-procedure)  
  (effected-component m-mission-component))
```

```
*****  
/*  
 * PANSYS: A Computer Model of Text Comprehension and Question  
 * Answering for Failure Analysis  
 */  
  
File: cases.mops  
/*  
 * Developed by: Sergio J. Alvarado  
 * Ronald K. Braun  
 * Kenrick J. Mock  
 */  
  
Artificial Intelligence Laboratory  
Computer Science Department  
University of California  
Davis, CA 95616  
/*  
 * Funds for the support of this study have been allocated by the  
 * NASA-Ames Research Center, Moffett Field, California, under  
 * Interchange No. NCA2-721.  
 */  
*****  
  
/* Domain knowledge representation for the DMS of the Space Station Freedom:  
 * Abstract representation of a case.  
 */  
  
(defmop m-case (m-root)  
  (failed-component m-oru FAILED)  
  (mode m-failure-mode  
    (failed-component m-oru FAILED))  
  (cause m-failure-cause  
    (failed-component m-oru FAILED))  
  (detection m-failure-detection  
    (failed-component m-oru FAILED))  
  (correction m-failure-correction  
    (failed-component m-oru FAILED))  
  (effect m-failure-effect))
```

```
*****  
;* FANSYS: A Computer Model of Text Comprehension and Question  
;* Answering for Failure Analysis  
;  
;* File: misc.mops  
;  
;* Developed by: Sergio J. Alvarado  
;* Ronald K. Braun  
;* Kenrick J. Mock  
;  
;* Artificial Intelligence Laboratory  
;* Computer Science Department  
;* University of California  
;* Davis, CA 95616  
;  
;* Funds for the support of this study have been allocated by the  
;* NASA-Ames Research Center, Moffett Field, California, under  
;* Interchange No. NC22-721.  
;  
;* Domain knowledge representation for the DMS of the Space Station Freedom:  
;* Miscellaneous mop defs.  
;  
(defmop m-question-constraint (m-processing-mop))
```

```

***** FANSYS: A Computer Model of Text Comprehension and Question
***** Answering for Failure Analysis *****

File: mem-structs.lsp
*****
Developed by: Sergio J. Alvarado
                Ronald K. Braun
                Kenrick J. Mock
*****
Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616
*****
Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Intercharge No. NCA2-721.
*****
This file defines the memory structures used to create the semantic
network of FANSYS' memory, as well as structures used in the
parsing process.
*****
structure for a MOP (a semantic node):
name:          the name of given mop
absts:         list of all immediate abstractions of this mop
all-absts:     list of all abstractions for this mop (in hierarchical
               order from spec to generalization)
specs:         list of specializations of this mop
slots:         list of slot and filler-type pairs which characterize
               this mop
type:          type of mop (instance or abstraction)
ips:           list of index patterns activated by this mop
assoc-fns:    maps-specific procedural information to guide processing
pms:           list of current predictions for this mop
ams:           list of activations for this mop in current parse
ip-refs:      list of index patterns which reference this mop
vars:          a list of variables that are active within the context of
               this mop
back:          a list of the mops which have slots that package this mop
equivalent:   a list of mops which are equivalent to this one, differing
               only with respect to their variable bindings
*****
Kolodner-style memory structures:
index-down:   list of downward pointers to specialization mops.
Each element in the list contains three elements,
an attribute name, a value (some exploded hierarchy),
and a sub-mop to point to. Allows traversal of hierarchy.
The format is ('attribute value mop-up/mop-down) ... )
index-up:     list of upward pointers to abstraction mops.
These are not currently used at this time in the
existing implementation of FANSYS.
*****
bad-index:   list of the attributes which are BAD Indices (i.e.,
unpredictive features) for a particular MOP.
These are used to indicate irrelevant indices
which should not be used for storage/traversal.
norms:        list of norms ((attribute value) ...) for the given
MOP. All sub-mops will share this generalization.
elaborations: list of possible elaborations (other attributes to
examine) for a given MOP. Not used in the existing
implementation of FANSYS.
*****
; marker definitions for prediction (PM) and activation (AM) markers
; (markers are implemented as free-floating organized structures)
;
; structure for an AM:
; parent: originator of the AM
; owners: list of all recipients of AM
; start: value of count when this am was created
; finish: in ips, value of count when the recognized concept is
; activated
;
; (destruct am parent owners start finish)
;
; structure for a PM
; parent: originator of the PM
;
; (destruct pm parent)
;
; mops are stored in a table for name to mop conversion
;
; mops are stored in a table for name to mop conversion
(setf *mops* nil)
(define-table name->mop (mop) *mops*)

```

```

;;;
;; name, abstraction, and an attribute/value pair, create the index
;; linking the abstraction to the specialization via index-down.
;; The abstraction gets the pair + spec mop, and the spec mop

;;
;; Currently, the mop-index-up index is NOT used. Consequently,
;; it has not been created to save memory and space (the expanded
;; indices can take a large amount of space).
;;

(defun Add-Index (specname abstname attribute value)
  (let ((spec (name->mop specname)) (abst (name->mop abstname)))
    (cond
      ((not (My-Member (list attribute value specname) (mop-index-down abst))))
       (setf (mop-index-down abst)
             (acons (list attribute value specname) (mop-index-down abst))))
      ; Index-up left out for now in the name of saving space
      ; (cond
      ;   ((not (My-Member (list attribute value abstname) (mop-index-up spec)))
      ;    (setf (mop-index-up spec)
      ;          (acons (list attribute value abstname) (mop-index-up spec)))))
      ;))

;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; My-Make-MOP
; Create a new MOP but leaves indices blank for now
; Name = Mop Name, Type = MOP or INSTANCE, slot = given slots
; New-MOP name nil type slots
;;

(defun My-Make-MOP (name type slots)
  (New-MOP name nil type slots)
)

;;
;; FANSYS Memory Module
;;
;; This code is loosely based upon the MicroMops code by Riesbeck
;; and Schank from Inside Case-Based Reasoning, but has incorporated
;; the memory indexing scheme proposed by Koldner in Computer Memory
;; Organization. This documentation focuses upon function definitions
;; and low-level implementation details. High level details are
;; available in the paper.

;; Modifications have been made to the ICBR Micromops code so that
;; attribute/value indices, initial generalization, and memory
;; retrieval are possible.

;; This version does not yet use "Find-Bad" to remove overspecialized
;; features automatically, or use "Find-Elaborations" to automatically
;; discover possible elaborations. These aspects are discussed in
;; the paper.

;;
;; Miscellaneous Development Comments:
;;
;; 4/6/93 Documentation and code cleanup
;; 10/29/92 Added hierarchical mods, very messy, needs documentation
;; sometime...also switched to structures rather than tables.
;; 12/4/93 This particular version uses (header mponame) as a slot..
;; This version also explodes MOP's out. A hash table has been
;; added to increase retrieval efficiency in re-computing the exploded
;; MOP's.

;;
;; 4/6/93 Function to add mop indices attribute/value pairs. Given a mop
;; one of the attribute/values given for a particular MOP.

;;
;; This is not a clear cut question. What determines if a attribute
;; matches another attribute? Should the two attributes match
;; completely, or allow for a partial match? If a partial match,
;
```

```

***** This particular function
***** how much should match, 50, 25, 75? This particular function
***** requires either an exact match, or that the HEADER slot of the
***** index in the MOP matches or the same as the given attribute
***** (i.e., the given attribute is identical to the index's value)
***** or if there is the percent match specified by CommonAbst.
*****
***** This returns a list of the attribute/mops which match.
*****
***** Is-Match-Down3 (mop attribute value)

(defun Is-Match-Down3 (mop attribute value)
  (let ((l nil))
    (for (i :in (mop-index-down (name->mop mop)))
      :WHEN (or (and (equal (car i) attribute) ; exact match
                      (equal (cadr i) value))
                  (and (listp (cadr i)) ; both lists
                      (listp value)
                      (equal (car i) attribute)
                      (CommonAbst (cadr i) value))
                  (> (MatchPercent value (cadr i) 't) 0.5))
      :AND (equal (car i) attribute)
            (equal value '*))
      :SAVE (list (caddr i) )))
  )

***** CommonAbst

(defun CommonAbst (values1 values2)
  (let ((vhead nil) (v2head nil))
    (cond
      ((and values1 values2)
       (setf vhead (assoc 'HEADER values1))
       (setf v2head (assoc 'HEADER values2)))
      ((MyAbstp (cadr v2head) (name->mop (cadr vhead))))
       (cond
         ((> (MatchPercent values2 values1 't) 0.7) 't)
         )
      )
    )
  )

***** Easy-Match-Down3

(defun Easy-Match-Down3 (mop attribute value)
  (let ((l nil))
    (for (i :in (mop-index-down (name->mop mop)))
      :WHEN (or (and (equal (car i) attribute) ; exact match
                      (equal (cadr i) value))
                  (and (listp (cadr i)) ; both lists
                      (listp value)
                      (equal (car i) attribute)
                      (CommonAbst2 (cadr i) value))
                  (> (MatchPercent value (cadr i) 't) 0.2))
      :AND (equal (car i) attribute)
            (and (equal (car i) attribute)
                 (equal (cadr i) attribute)
                 (and (equal (cadr i) attribute)
                      (equal (caddr i) attribute)
                      (equal (cadr i) (caddr i))))))
  )

***** Is-Match-Up

(defmacro Is-Match-Up2 (mop attribute value)
  '(Is-Match-Up2 ',mop ',attribute ',value))

***** CommonAbst

(defun CommonAbst (values1 values2)
  (let ((vhead nil) (v2head nil))
    (cond
      ((and values1 values2)
       (setf vhead (assoc 'HEADER values1))
       (setf v2head (assoc 'HEADER values2)))
      ((equal (car (mop-absts (name->mop (cadr vhead)))) (car (mop-absts (name->mop (cadr v2head))))))
       (cond
         ((> (MatchPercent values2 values1 't) 0.7) 't)))
      )
    )
  )

```

```

(defun Is-Match-Up2 (mop attribute value)
  (let ((l 0))
    (for (i :in (mop-index-up (name->mop mop)))
      :WHEN (and (equal (car i) attribute)
                  (equal (cadr i) value))
        :FIRST (caddr i)))
  *****)

***** Get-Values-For-Attrib
***** Gets all values that match some given attribute
***** for a given MOP attribute/value indices. In other words, if
***** we are given a MOP and the attribute "REASON" then this returns
***** a list of all values for MOP where the attribute is REASON;
***** e.g., the value might be "BAD-DISK" or something of this sort.
***** This is used for elaboration to allow search to proceed along an
***** additional path.
***** 

(defun Get-Values-For-Attrib (mop attribute)
  (let ((l 0))
    (for (i :in (mop-index-down (name->mop mop)))
      :WHEN (equal (car i) attribute)
        :SAVE (caddr i)))
  *****)

***** Transform-attribute-reqr
***** This auxiliary function takes an input query (attribute-reqr list)
***** and transforms it into a form for processing. It basically
***** replaces equals with Is-Match-Down and inserts the mop-name.
***** This is necessary for processing, since we want to keep track
***** of what matches. Regular "equals" won't keep track of what
***** matches for us.
***** 

***** Given some attribute-rear list in the form:
***** (equal computer macintosh) (equal keyboard extended)
***** (and (equal computer macintosh) (equal keyboard extended))
***** (or (equal computer macintosh) (equal keyboard extended))
***** Turn this into the implementation form of:
***** (Is-Match-Down m-root computer macintosh)
***** (All-Same (Is-Match-Down m-root computer macintosh)
***** (Is-Match-Down m-root keyboard extended))
***** (or (Is-Match-Down m-root computer macintosh)
***** (Is-Match-Down m-root keyboard extended))
***** As a result, we can just "eval" what is returned to see if
***** there is a match or not.
***** 

***** This function is basically just a pattern search/replacement
***** routine.
***** 

(defun transform-attribute-reqr (mop attribute-reqr)
  (let ((newlist nil))
    (for (i :in attribute-reqr)
      :DO (cond
            ((listp i)
              (setf newlist
                    (append newlist (list i))))
            (t
              (setf newlist (append newlist (list i)))))

  ***** easy-transform-attribute-reqr
***** This version is identical to the previous, except it uses
***** easy-match-down. Used for question/answering.
***** 

(defun easy-transform-attribute-reqr (mop attribute-reqr)
  (let ((newlist nil))
    (for (i :in attribute-reqr)
      :DO (cond
            ((listp i)
              (setf newlist
                    (append newlist
                          (list (easy-transform-attribute-reqr mop i)))))

            (t
              (or (equal i 'equal) (equal i 'eq))
                (setf newlist
                      (append newlist
                            (list 'Easy-Match-Down mop))))))

  ***** (equal i 'and)
***** (setf newlist
*****   (append newlist (list 'All-Same)))
***** (equal i 'or)
***** (setf newlist
*****   (append newlist (list 'All-Or)))
***** (t
*****   (setf newlist (append newlist (list i)))))

  ***** All-Same
***** This just makes sure everything in l is the same thing. It's like
***** a big equals statement; all items in the list must be equal to be
***** true. Used for checking if indices match.
***** 

(defun All-Same (lrest l)
  (if (null l)
    (cond
      ((equal (length l) 1) (car l))
      ((equal (car l) (cadr l))
        (All-Same2 (cdr l)))
      (t nil)))

  ***** All-Or
***** Do an OR on things in l. However, unlike a regular OR, this OR's
***** everything, and doesn't stop when the first true is found. This
***** is so we can generate a list of potential mops to visit, stored
***** in the global variable *TEMPLIST*.
***** 

(defun All-Or (l)
  (setf newlist
        (append newlist (list (transform-attribute-reqr mop l)))))

  (or (equal i 'equal) (equal i 'eq))
    (setf newlist
          (append newlist
                (list 'Is-Match-Down mop)))))

  (append newlist (list 'TEMPLIST)).

```

;; Note that this implementation is not totally correct if we have OR's
 ;;; imbedded within AND's.

(defun All-Or (&rest l)
 (All-Or2 l nil))

(defun All-Or2 (l found)
 (cond
 ((null l) *TEMPLIST*)
 ((not (equal nil (car l)))
 (for (i :in (car l))
 :do (setf *TEMPLIST* (append *TEMPLIST* (list (cadr i))))
 (setf found (car l))
 (All-Or2 (cdr l) *TEMPLIST*))
 (t (All-Or2 (cdr l) *TEMPLIST*))))

 General-Info

Return list of general info (norms) of the mop-found
 ;;; By merely traversing all link upwards in the abstraction hierarchy
 ;;; This can be used if one would like to see all generalizations of
 ;;; some MOP which has been returned.

(defun General-Info (curmop &optional listsofar)
 (cond
 ((equal curmop 'M-ROOT) nil)
 (t
 (setf listsofar (append listsofar (mop-norms (name->mop curmop))))
 (for (i :in (mop-index-up (name->mop curmop)))
 :do (setf listsofar
 (append listsofar (general-info (caddr i))))
 listsofar))

 General-Info2

Give general norm info, but not as general as above function.
 ;;; Only return norms for those mops along path that was
 ;;; traversed rather than everything above it. The mops which were
 ;;; traversed must be passed in via the moplisp parameter.
 ;;; This merely limits the output; if there are a large number of
 ;;; abstractions, than General-Info might return a lot of stuff.
 ;;; This will only return more pertinent information.

(defun General-Info2 (curmop moplisp &optional normlist)
 (cond
 ((equal curmop 'M-ROOT) nil)
 (t
 (setf normlist (append normlist (mop-norms (name->mop curmop))))
 (for (i :in (mop-index-up (name->mop curmop)))
 :do
 (cond
 (my-member (caddr i) moplisp)
 (setf normlist
 (append normlist (general-info2 (caddr i) moplisp)))))))

normlist)

 MOP-Search1

Search MOPs for match using only given attribute/values.
 ;;; This is searching for a direct match using no elaboration.
 ;;; Elaboration is done at a level higher than this; to elaborate,
 ;;; additional attributes will be appended onto the query before
 ;;; this function is invoked.

First, the input query is transformed into a workable format
 ;;; which can be used with eval.

Next, this function works by employing a depth-first recursive
 ;;; search along matching indices only. An index is considered
 ;;; matching if its attribute/value pair satisfies the expression
 ;;; given in attribute-reqr (the input query). Whenever a MOP
 ;;; matches, the name of that MOP is stored in the global variable
 ;;; *TEMPLIST* by the All-Or or All-Same functions. This is used
 ;;; to construct the foundlist parameter, which is ultimately passed
 ;;; back. It will contain the names of all the MOPs traversed
 ;;; to reach an instance MOP, and the names of all instance MOPs reached.

If no instance MOP is reached, NIL is returned.

(defun MOP-Search1 (mopname attribute-reqr &optional foundlist)
 (let ((mop (name->mop mopname))
 (foundmop nil)
 (moplisp nil)
 (alist nil)
 (tempacket nil)
 (query nil))
 (and *verbose* (st (format nil "Current MOP: ~S ~ mopname)))
 (setf *TEMPLIST* nil)
 (setf query (easy-transform-attribute-reqr mopname attribute-reqr))
 (setf tempacket (eval query)) ; Identify mops to recurse on
 (setf foundlist (append foundlist (list mopname)))
 (cond
 ((instance-mopp mop) foundlist)
 ((null foundmop) nil))
 (t
 (and *verbose*
 (st (format nil " Input matches Indices to: -S-~S~S moplisp)))
 (for (i :in moplisp)
 :when (not (null (setf alist
 (MOP-Search1 i attribute-reqr foundlist))))
 :splice alist)))

 Miscellaneous Functions

The following are small little utility functions which I use for
 ;;; searching and removing duplicates.
 ;;; Remove-Duplicates and NIL from a list
 ;;; searching and removing duplicates.

(defun Remove-Dupe-Nil (l)

```

;;;
;;; elaboration.
;;;
;;; The current implementation of FANSYS does not add elaborations,
;;; but this has been partially implemented for future expansion for
;;;
;;; allowing the learning of elaborations.
;;;

(defun Add-elaboration (match-values elaboration attribute-reqr)
  (let ((new-query ' (or)))
    (cond
      ((equal match-values nil)
       attribute-reqr)
      (t
       (for (l :in match-values)
            :DO
              (setf new-query (append new-query
                                      (list (list 'equal elaboration l))))))
        (append new-query (list attribute-reqr)))))

***** Best-Match *****

(defun My-Remove-Dupe (l)
  (let ((newlist nil)
        (for (l :in l)
             :DO
               (cond
                 ((equal (My-Member l newlist) nil)
                  (setf newlist (append newlist (list l)))))))
    newlist))

; Remove item from a list.
; =====

(defun My-Replace (item 1)
  (for (l :in l)
       :WHEN (not (equal l item))
       :SAVE 1)
  :SAVE 1)

; Replace item from a list with new.
; =====

(defun My-RePlace (item newitem 1)
  (for (l :in l)
       :SAVE (cond
                  ((equal l item) newitem)
                  (t l)))
  :WHEN (not (equal nil (My-Member l 12)))
  :DO (setq c (+ c 1)))

; count number of non-matches, membership of item in a list
; =====

(defun countit (l1 12)
  (let ((c 0))
    (for (l :in l1)
         :WHEN (not (equal nil (My-Member l 12)))
         :DO (setq c (+ c 1)))
  c))

; Add-elaboration
; =====

(defun My-Member (item 1)
  (for (l :in l)
       :WHEN (equal l item)
       :FIRST 1))

; Build query with elaboration (additional attribute/value indices).
; =====

(defun My-Max (mplist)
  (cond
    ((equal 'MOP (mop-type (name->mop 1)))
     (setf mval (append mval (list '0)))
     (equal 'INSTANCE (mop-type (name->mop 1)))
     (setf caseslots (GetExplodedMop 1))
     (setf mval (append mval
                        (list (MatchPercent newlist caseslots))))))
    (t
     (list (nth (largest-pos mval) mplist) (My-Max mval)))))

***** My-Max *****

; One of a number of additional small utility-type functions.
; =====

; Return largest element in a list. Used to figure out the
; best matching MOP.
; =====

(defun My-Max (l)
  ; Add-elaboration
  ; =====
  ; Build query with elaboration (additional attribute/value indices).
  ; =====
  ; This takes the current query, given in attribute-reqr, and just
  ; adds an OR onto it:
  ; =====
  ; Current-Query + Elaboration/Match-value=>(or (Current-Query)
  ; (eq Elaboration Match-Value))
  ; =====

```

```

let (biggest 0)
  (for (i :in l)
    :do (cond ((< biggest i) (setf biggest i)))
      ;;; Used to determine the best matching MOP.
      ;;; Count # of matches between i1 and i2 and return it.
      ;;; Count-Matches
      ;;; Used to determine the best matching MOP.
      ;;; Count-Matches

(defun count-matches (i1 i2)
  (cond
    ((equal i1 nil) 0)
    ((my-member (car i1) i2)
     (+ 1 (count-matches (cdr i1) i2)))
    (t (count-matches (cdr i1) i2)))))

(defun largest-pos ()
  ; Return position of largest number in a list..Used to
  ; determine the best matching MOP.
  (let ((lv 0)
        (maxpos 0)
        (pos 0))
    (for (i :in l)
      :do
        (cond
          ((>= lv i)
           (setf pos (+ 1 pos)))
          ((< lv i)
           (setf maxpos pos)
           (setf pos (+ 1 pos))
           (setf lv i)))
        maxpos)))
  ;;; Used from add-instance to make sure a MOP isn't already in memory.
  ;;; Used from add-instance to make sure a MOP isn't already in memory.

(defun pairs-query (pairlist)
  ;;; Pairs-Query
  (let ((newlist '(or)))
    (for (i :in pairlist)
      :do (setf newlist (append newlist (list (append ' (equal 1)))))))
  ;;; Used from add-instance to make sure a MOP isn't already in memory.
  ;;; Used from add-instance to make sure a MOP isn't already in memory.

(defun integrated-add-instance
  ;;; Integrated-Add-Instance
  ;;; This is the function to add a new mop to memory.
  ;;; Once again, this is used in determining the best match.
  ;;; Given a query (e.g. (or (equal computer ibm) ... )), this function
  ;;; extracts the attributes so that they may be compared with the
  ;;; slots of a particular MOP. In the above case, we'd get back
  ;;; ((computer ibm) ... )
  ;;; Extract attributes and values from given query.
  ;;; This works by recursively going through the list and just building
  ;;; up the "newlist" list, depending on whether or not the current
  ;;; element is a list starting with (or, and), a list with (equal ..)
  ;;; or an element.

  ;;; Add-Instance does a little bookkeeping and then
  ;;; calls Add-Instance2 to do all of the real work.
  ;;; The bookkeeping involves exploding out the MOP and adding this
  ;;; module, hence the name "Integrated-Add-Instance".
  ;;; This can also check to see if a duplicate case already exists
  ;;; in memory; this feature is currently disabled to speed the memory
  ;;; creation process.
  ;;; (and (listp l)
  ;;;   (defun match-attribute-list (attribute-reqr)
  ;;;     (let ((newlist nil)
  ;;;       (for (i :in attribute-reqr)
  ;;;         :do (cond
  ;;;           ((or (equal (car i) 'or) (equal (car i) 'and)))
  ;;;             (setf newlist (append newlist (match-attribute-list i))))
  ;;;           ((listp i)
  ;;;             (setf newlist
  ;;;               (append newlist (list (match-attribute-list i)))))
  ;;;             ((not (or (equal i 'equal) (equal i 'eq) (equal i 'or)
  ;;;             (equal i 'and)))
  ;;;               (setf newlist (append newlist (list i))))))
  ;;;             newlist)))
  ;;;   ;;; Select-Indices
  ;;;   ;;; Given a list of features, select those to use as indices. Right
  ;;;   ;;; now it weeds out pre-selected features. The list "l" must
  ;;;   ;;; be of the form ((attribute value) (attribute value) ...)
  ;;;   ;;; The features that are weeded out are any that match the slot
  ;;;   ;;; "bad-index" for that mop. These "bad-indices" are determined when
  ;;;   ;;; adding instances to memory, and it is done using a threshold
  ;;;   ;;; analysis.
  ;;;   ;;; In the current implementation, some bad indices may be hand-coded,
  ;;;   ;;; e.g., manually set mop-bad-index to contain "failure cause" since
  ;;;   ;;; all cases share almost all the same failure causes and this will
  ;;;   ;;; be non-predictive.
  ;;;   ;;; In the current implementation, some bad indices may be hand-coded.
  ;;;   ;;; Right
  ;;;   ;;; Given a list of features, select those to use as indices. Right
  ;;;   ;;; now it weeds out pre-selected features. The list "l" must
  ;;;   ;;; be of the form ((attribute value) (attribute value) ...)
  ;;;   ;;; The features that are weeded out are any that match the slot
  ;;;   ;;; "bad-index" for that mop. These "bad-indices" are determined when
  ;;;   ;;; adding instances to memory, and it is done using a threshold
  ;;;   ;;; analysis.
```

```

; This function is called after a new MOP has already been created
; during the parsing process and when the MOP is ready to be added
; to the attribute/value indexing hierarchy. It is called with
; the list of attribute/value features and the name of the new
; MOP (not the structure itself).
; *****

(defun Integrated-Add-Instance (features mophname meta-parent)
  (cond
    ((equal *mode* 'case)
     (let ((found nil) (temp nil))
       (don't search for adding same duplicate mop...yet...it's slow
       ; don't search for adding same duplicate mop...yet...it's slow
       ; setf found (MOP-Search1 meta-parent (pairs-query features)))
       (setf temp (explode-it mophname))
       (setf temp (reformat temp))
       ; Store exploded case in hash table for fast future retrieval
       (setf (gethash mophname *Exploded*) temp)
       (cond
         ((or (equal found nil)
               (not (equal (cadr (best-match found (pairs-query temp)))
                           1)))
           (and *verbose*
                (st (format nil "Adding with meta-parent: ~S~" meta-parent)))
           ; Call Add-Instance2 to do all the work
           (Add-Instance2 mophname meta-parent temp)
           (and *verbose* (st (format nil "Finished Add...~S~" )))
           (t
             (princ "MOP already in memory.")
             (terpri)))))))
    ;*****)

; Add-Instance2
; Performs most of the work in adding a new MOP case to memory.
; This entails traversing the network to see where we should put
; the new case, and finally putting the case there with the new
; indices.

1. Select:
; Add new MOP to memory. This picks a name and then does
; recursive addition. The current strategy to add a new MOP is:
; 1. If direct conflict with new features to norms, remove norm
; 2. Initialize set S to contain all new indices
; 3. Search for feature to use when elaborating
; 4. Create new Instance-MOP with slots
; 5. Start at the root:
;   A) Features to use as indices
;   B) Possibly remove features as bad indices
;   C) Search for feature to use when elaborating
; 2. Create new M-MOP
; 3. If index doesn't exist, create index between root and
;    the new MOP. Remove index from S.
; 4. Else if index exists to Sub-MOP, recurse with
;    the new MOP as the root (step 3)
; 5. Else if index exists to Instance-MOP then:
;    Create new M-MOP
;    Calculate similarities between instance, new mop.
;    Add similarities as norms to new M-MOP.
;    Calculate differences between instances, new mops.
;    Index MOPs based on differences.
;    Remove index traversed from set of new indices.
;*****)

(defun Add-Instance2 (new-mop cur-mop features)

```

```

(let ((newmop nil)
      (submop nil)
      (packet nil)
      (prefeatures nil)
      (slimfeatures nil)
      (foundval nil)
      (foundmop nil))
  (Doesn't weed out bad mops or do elaboration at this point yet
  ; However, we do use FindMultiple (see comments below).
  ; (Find-Bad cur-mop)
  ; (Find-Elaborations (name->mop cur-mop) features)
  ; (setf features (select-indices (name->mop cur-mop)))
  ; Make sure norms are consistent by just finding all similarities
  ; and setting the norms to the similarities between the old norms
  ; and the new case. As a future modification we shouldn't just
  ; overwrite the old norms, but should remember what was once a
  ; generalization
  ; (setf (mop-norms (name->mop cur-mop))
  ;       (mop-norms (name->mop cur-mop)))
  ; Currently, only traverses one link when adding. Should
  ; we traverse all matches? Will there ever be a match with more than
  ; one link? If more links are traversed, there will be many more
  ; indices created.
  ; Slimfeatures - removes each attribute/value as it goes down the
  ; hierarchy to reduce the amount of indices which are added. For
  ; example, if we have the features (computer ibm) (color blue) then
  ; if we add 'computer' then remove it from the set of features so
  ; it isn't indexed again. This dramatically cuts down the number of
  ; indices which are created (and prevents exponential explosion).
  ; (setf slimfeatures features)
  ; (for (l1 in features)
  ;       (do
  ;         (cond
  ;           ((and (find-slimfeature (my-remove l1 slimfeatures))
  ;                 (not (equal (car (match-down3 cur-mop (car l1))))))
  ;                  (not (equal (setf packet (car
  ;                                         (Is-Match-Down3 cur-mop (car l1))))))
  ;                         (nil))
  ;                         (not (equal nil (setf foundmop (cadr packet)))))
  ;                         (not (equal nil (setf foundval (car packet))))))
  ;             (princ "i=~")
  ;             (st (format nil "~<Foundmop:~S~" foundmop)))
  ;             (and *verbose* (st (format nil "~<Foundmop:~S~" foundmop)))
  ;             (cond
  ;               ((equal (mop-type (name->mop foundmop)) 'MOP)
  ;                (Add-Instance2 new-mop foundmop slimfeatures)
  ;                (cond (null (My-Member (car l1) prefeatures)
  ;                      (setf prefeatures (cons (car l1) prefeatures)
  ;                            )))
  ;                     ((equal (mop-type (name->mop foundmop)) 'INSTANCE)
  ;                      (make-submop foundmop)
  ;                      (cond
  ;                        ((equal foundmop new-mop)
  ;                         (add-index new-mop cur-mop (car l1) (cadr l1)))
  ;                         (t
  ;                           (setf submop (spec-name 'comp) 'mop)
  ;                           (My-Make-Mop submop 'mop nil)
  ;                           (set bad indices. These will be computationally expensive if there
  ;                                are a lot of the same type of index...)))
  ;                     )))))
  ;           )))))
  ;*****)

;*****)

```

```

;;; Occur over 4 times.
*****  

(setf (mop-bad-index (name->mop submop))
      (FindMultiple features))  

;;; Inherit parent's bad indices  

(setf (mop-bad-index (name->mop submop))
      (append (mop-bad-index (name->mop cur-mop))
              (mop-bad-index (name->mop submop))))  

;;; Add past indices as bad  

(setf (mop-bad-index (name->mop submop))
      (append (mop-bad-index (name->mop submop))
              prefeatures))  

;;; Delete any possible duplicates
(setf (mop-bad-index (name->mop submop))
      (My-Remove-Dupe (mop-bad-index (name->mop submop))))  

;;; Remove old indices
(setf (mop-index-down (name->mop cur-mop)
                      (my-remove (list (car 1) foundval) foundmop)
                      (mop-index-down (name->mop cur-mop)))  

      (Upward indices out for now to save memory and space
       (setf (mop-index-up (name->mop foundmop)
                           (list (car 1) foundval cur-mop)
                           (mop-index-up (name->mop foundmop))))  

;;; Add new index
(cond
  ((and (listp (cadr 1)) (listp foundval)) ;;; both lists
   (add-index submop cur-mop (car 1)
             (cadar (Find-Sim-Recursive
                     (list 1)))))  

  ((t (add-index submop cur-mop (car 1) (foundval))))  

;;; Add similarities as norms
  (setf (mop-norms (name->mop submop))
        (Find-Sim-Recursive
         (GetExplodedMop foundmop)
         (GetExplodedMop new-mop)))  

;;; Add Index as norm
  (setf (mop-norms (name->mop submop)
                    (my-remove-dupe (mop-norms (name->mop submop))))  

;;; Add indices
  (Multiple-Indices submop new-mop
    (Find-Dif-Recursive features
      (select-indices (name->mop submop)
                     (GetExplodedMop foundmop))))  

  (Multiple-Indices submop foundmop
    (Find-Dif-Recursive
     (select-indices (name->mop submop)
                   (GetExplodedMop foundmop)
                   features)))
  (cond ((null (My-Member (car 1) prefeatures))
         (setf prefeatures (cons (car 1) prefeatures))))  

  ))))  

(t ;;; Otherwise, no existing index so add it
  (add-index new-mop cur-mop (car 1) (cdr 1))))  

*****  

;;; Findmultiple
;;; Find any multiple slots with same attribute, and add these
;;; to the bad index so we only traverse them once..Right now
;;; bad ones are those that occur at least 4 times. This number
;;; has been derived somewhat ad-hoc, by examining the domain
;;; beforehand and noticing that poor indices are those which
*****  

(defun FindMultiple (features)
  (let ((foundlist nil) (badlist nil))
    (for (i :in features)
      (DO (cond ((not (My-Member (car 1) foundlist))
                 (setf foundlist (cons (car 1) foundlist)))
                ((not (My-Member (car 1) badlist))
                 (setf badlist (cons (car 1) badlist))))  

        )  

      (for (i :in badlist)
        (WHEN (>= (CountEmup 1 features) 4)
          :SAVE 1))  

    (defun CountEmup (item features)
      (let ((num 0))
        (for (i :in features)
          (DO (cond ((equal (car 1) item) (setf num (+ 1 num)))
                    num))  

        ))  

      ))  

      ;;; Remove-From-Norm
      ;;; Given a set of norms and features, return only those features
      ;;; which differ from the norm
      ;;;  

      (defun Remove-From-Norm (norms features)
        (for (i :in features)
          (WHEN (for (j :in norms)
                    :WHEN (equal 1 j)
                    :SAVE 1)
            nil)
          :SAVE 1))  

      ))  

      ;;; Multiple-Indices
      ;;; This function adds indices between Mops.
      ;;; Add indices given in index-list pointing from submop to new-mop.
      ;;; attribute/value indices.
      (defun Multiple-Indices (submop new-mop index-list)
        (for (i :in index-list)
          (DO (add-index new-mop submop (car 1) (cadr i))))  

      ))  

      ;;; My-DAH
      ;;; Function to show my abstraction hierarchy, along with the
      ;;; attribute/value indices.
      (defun My-DAH (root optional attrib value numblanks)
        (progn
          (cond ((equal numblanks nil) (setf numblanks 0)))
          (PrintSpace numblanks)
          (princ "MOP:")
          (princ root)))
      ))  


```

```

: DO
  (princ " Attrib:")
  (princ attrib)
  ; (princ " Value:")
  ; (princ value)
  (terpri)
  (for (l :in (mop-index-down (name->mop root)))
    :DO
      (My-DAH (caddr l) (car l) (cadr l) (+ numblanks 2)) t)
  ;*****+
  ;** PrintSpace
  ;** Used for printing spaces in my abs hierarchy for nice formatting
  ;*****+
  (defun PrintSpace (num)
    (cond
      ((> num 0)
       (princ " ")
       (PrintSpace (- num 1)))))

  ;** Get-Index-Attributes
  ;** Returns a list of all attributes used for indexing down from
  ;** a given mop. This is used in the "Find-Bad" function.
  ;*****+
  (defun Get-Index-Attributes (mop)
    (let ((l nil))
      (for (l :in (mop-index-down mop))
        :DO
          (cond
            ((not (my-member (car l) l))
             (setf l (append (list (car l)) l)))
            1)))

  ;** Find-Bad
  ;** Not currently implemented - what follows is a potential scheme
  ;** which may be used for future expansion.

  ;** Finds "Bad" indices for Mops. In this case, "Bad" means the index
  ;** is over-specialized (the other case is a under-specialized index)
  ;** which is removed through secondary generalization.

  ;** This checks if the ratio of instances/mops for a given attribute
  ;** index is too high (>2 and at least 10 instances seen so far).
  ;** If so, save this attribute as an overspecialization which SHOULDN'T
  ;** be used for indexing by storing it under "mop-bad-index" for the
  ;** given mop. Then, go and wipe out any sub-mops indexed below this
  ;** bad index, and remove any links to other mops under this index.
  ;*****+
  (defun Find-Bad (mop)
    (let ((attribs nil)
          (values nil)
          (mops 0)
          (insts 0)
          (setf attribs (Get-Index-Attributes (name->mop mop)))
          (for (l :in attribs)

```



```

***** added to memory. *****
Query= ((computer (type mac)
                   (color green)
                   (size big)
                   (taste sweet))
          (location school)
          (time past))

When comparing the query against the case, these values are
computed from the leaves and propagated back up the hierarchy:
((computer (type 0)
            (color 0)
            (size 1)
            (taste 0)
            (location 1)
            (time 0))
 these propagate up:
            ((computer 1/4)
             (location 1)
             (time 0))
and the final match percentage is given as 1.25/3 or 45%.
Notice that the final value may or may not be indicative of a
really good match if the features which are in use are not
relevant or if some features are more important than others.
Weights may be necessary in order to weight the relevant features
more importantly than the others.

***** Use "HeadMatch" flag to allow matching of headers to determine match.
***** MatchPercent (query case &optional HeadMatch)
(let ((totl 0) (nummatches 0) (percent 0) (value nil) (submatchval 0)
      (ghead nil) (thead nil) (largest 0)
      (multmatch nil))
  (setq totl (length query))
  (setf qhead (assoc 'HEADER query))
  (setf chead (assoc 'HEADER case))
  (cond
    ; If flag & header of query is abst of index header
    ((and (null HeadMatch)
          (not (null qhead)) (not (null chead))
          (myabstp (cadr qhead) (name->mop (cadr chead))))
     1)
    ; say it matches 100% right off the bat
    (t
      (for (i :in query)
        (do
          ((setq value (Same-Attrib (car i) case)) ; Attr Match
           (setf largest 0) (setf multmatch nil)
           (do ((theplist (cdr i nil))
                ((or (null theplist) (equal largest 1)))
                (setq j (car theplist))
                (setq theplist (cdr theplist))
                (cond
                  ((equal (car i) 'HEADER) ; dont make headers match for now
                   ; setq largest 1)
                  ((equal (cadr i) 'HEADER) ; dont make headers match for now
                   ; setq largest 1)
                  ((equal (cadr i) 'value)
                   (setf largest 0) (setf multmatch nil)
                   (do ((theplist (cdr i nil))
                        ((or (null theplist) (equal largest 1)))
                        (setq j (car theplist))
                        (setq theplist (cdr theplist))
                        (cond
                          ((equal (car i) 'HEADER) ; dont make headers match for now
                           ; setq largest 1)
                          ((equal (cadr i) 'value)
                           (setf largest 0) (setf multmatch nil)
                           (do ((theplist (cdr i nil))
                                ((or (null theplist) (equal largest 1)))
                                (setq j (car theplist))
                                (setq theplist (cdr theplist))
                                (cond
                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                   ; setq largest 1)
                                  ((equal (cadr i) 'value)
                                   (setf largest 0) (setf multmatch nil)
                                   (do ((theplist (cdr i nil))
                                        ((or (null theplist) (equal largest 1)))
                                        (setq j (car theplist))
                                        (setq theplist (cdr theplist))
                                        (cond
                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                           ; setq largest 1)
                                          ((equal (cadr i) 'value)
                                           (setf largest 0) (setf multmatch nil)
                                           (do ((theplist (cdr i nil))
                                                ((or (null theplist) (equal largest 1)))
                                                (setq j (car theplist))
                                                (setq theplist (cdr theplist))
                                                (cond
                                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                                   ; setq largest 1)
                                                  ((equal (cadr i) 'value)
                                                   (setf largest 0) (setf multmatch nil)
                                                   (do ((theplist (cdr i nil))
                                                        ((or (null theplist) (equal largest 1)))
                                                        (setq j (car theplist))
                                                        (setq theplist (cdr theplist))
                                                        (cond
                                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                                           ; setq largest 1)
                                                          ((equal (cadr i) 'value)
                                                           (setf largest 0) (setf multmatch nil)
                                                           (do ((theplist (cdr i nil))
                                                                ((or (null theplist) (equal largest 1)))
                                                                (setq j (car theplist))
                                                                (setq theplist (cdr theplist))
                                                                (cond
                                                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                   ; setq largest 1)
                                                                  ((equal (cadr i) 'value)
                                                                   (setf largest 0) (setf multmatch nil)
                                                                   (do ((theplist (cdr i nil))
                                                                        ((or (null theplist) (equal largest 1)))
                                                                        (setq j (car theplist))
                                                                        (setq theplist (cdr theplist))
                                                                        (cond
                                                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                           ; setq largest 1)
                                                                          ((equal (cadr i) 'value)
                                                                           (setf largest 0) (setf multmatch nil)
                                                                           (do ((theplist (cdr i nil))
                                                                                ((or (null theplist) (equal largest 1)))
                                                                                (setq j (car theplist))
                                                                                (setq theplist (cdr theplist))
                                                                                (cond
                                                                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                   ; setq largest 1)
                                                                                  ((equal (cadr i) 'value)
                                                                                   (setf largest 0) (setf multmatch nil)
                                                                                   (do ((theplist (cdr i nil))
                                                                                        ((or (null theplist) (equal largest 1)))
                                                                                        (setq j (car theplist))
                                                                                        (setq theplist (cdr theplist))
                                                                                        (cond
                                                                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                           ; setq largest 1)
                                                                                          ((equal (cadr i) 'value)
                                                                                           (setf largest 0) (setf multmatch nil)
                                                                                           (do ((theplist (cdr i nil))
                                                                                                ((or (null theplist) (equal largest 1)))
                                                                                                (setq j (car theplist))
                                                                                                (setq theplist (cdr theplist))
                                                                                                (cond
                                                                                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                                   ; setq largest 1)
                                                                                                  ((equal (cadr i) 'value)
                                                                                                   (setf largest 0) (setf multmatch nil)
                                                                                                   (do ((theplist (cdr i nil))
                                                                                                        ((or (null theplist) (equal largest 1)))
                                                                                                        (setq j (car theplist))
                                                                                                        (setq theplist (cdr theplist))
                                                                                                        (cond
                                                                                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                                           ; setq largest 1)
                                                                                                          ((equal (cadr i) 'value)
                                                                                                           (setf largest 0) (setf multmatch nil)
                                                                                                           (do ((theplist (cdr i nil))
                                                                                                                ((or (null theplist) (equal largest 1)))
                                                                                                                (setq j (car theplist))
                                                                                                                (setq theplist (cdr theplist))
                                                                                                                (cond
                                                                                                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                                                   ; setq largest 1)
                                                                                                                   ((equal (cadr i) 'value)
                                                                                                                   (setf largest 0) (setf multmatch nil)
                                                                                                                   (do ((theplist (cdr i nil))
                                                                                                                        ((or (null theplist) (equal largest 1)))
                                                                                                                        (setq j (car theplist))
                                                                                                                        (setq theplist (cdr theplist))
                                                                                                                        (cond
                                                                                                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                                              
***** This important function determines how closely some expanded
representation matches another representation. This is done in
the following manner:
- For each hierarchy level: If there are X attribute/value
  pairs in the query, and Y attribute/value pairs in the case,
  compute how many X's match the Y's. If Z pairs from X's match
  out of the Y's then return Z/Y as the answer.
- Do the above scheme to compute a value for each attribute
  until we reach a terminal set.
Example: Case = ((computer (type ibm)
                           (color blue)
                           (size big))
                  (location school))

```

```

***** Query= ((computer (type mac)
                   (color green)
                   (size big)
                   (taste sweet))
          (location school)
          (time past))

When comparing the query against the case, these values are
computed from the leaves and propagated back up the hierarchy:
((computer (type 0)
            (color 0)
            (size 1)
            (taste 0)
            (location 1)
            (time 0))
 these propagate up:
            ((computer 1/4)
             (location 1)
             (time 0))
and the final match percentage is given as 1.25/3 or 45%.
Notice that the final value may or may not be indicative of a
really good match if the features which are in use are not
relevant or if some features are more important than others.
Weights may be necessary in order to weight the relevant features
more importantly than the others.

***** Use "HeadMatch" flag to allow matching of headers to determine match.
***** MatchPercent (query case &optional HeadMatch)
(let ((totl 0) (nummatches 0) (percent 0) (value nil) (submatchval 0)
      (ghead nil) (thead nil) (largest 0)
      (multmatch nil))
  (setq totl (length query))
  (setf qhead (assoc 'HEADER query))
  (setf chead (assoc 'HEADER case))
  (cond
    ; If flag & header of query is abst of index header
    ((and (null HeadMatch)
          (not (null qhead)) (not (null chead))
          (myabstp (cadr qhead) (name->mop (cadr chead))))
     1)
    ; say it matches 100% right off the bat
    (t
      (for (i :in query)
        (do
          ((setq value (Same-Attrib (car i) case)) ; Attr Match
           (setf largest 0) (setf multmatch nil)
           (do ((theplist (cdr i nil))
                ((or (null theplist) (equal largest 1)))
                (setq j (car theplist))
                (setq theplist (cdr theplist))
                (cond
                  ((equal (car i) 'HEADER) ; dont make headers match for now
                   ; setq largest 1)
                  ((equal (cadr i) 'HEADER) ; dont make headers match for now
                   ; setq largest 1)
                  ((equal (cadr i) 'value)
                   (setf largest 0) (setf multmatch nil)
                   (do ((theplist (cdr i nil))
                        ((or (null theplist) (equal largest 1)))
                        (setq j (car theplist))
                        (setq theplist (cdr theplist))
                        (cond
                          ((equal (car i) 'HEADER) ; dont make headers match for now
                           ; setq largest 1)
                          ((equal (cadr i) 'value)
                           (setf largest 0) (setf multmatch nil)
                           (do ((theplist (cdr i nil))
                                ((or (null theplist) (equal largest 1)))
                                (setq j (car theplist))
                                (setq theplist (cdr theplist))
                                (cond
                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                   ; setq largest 1)
                                  ((equal (cadr i) 'value)
                                   (setf largest 0) (setf multmatch nil)
                                   (do ((theplist (cdr i nil))
                                        ((or (null theplist) (equal largest 1)))
                                        (setq j (car theplist))
                                        (setq theplist (cdr theplist))
                                        (cond
                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                           ; setq largest 1)
                                          ((equal (cadr i) 'value)
                                           (setf largest 0) (setf multmatch nil)
                                           (do ((theplist (cdr i nil))
                                                ((or (null theplist) (equal largest 1)))
                                                (setq j (car theplist))
                                                (setq theplist (cdr theplist))
                                                (cond
                                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                                   ; setq largest 1)
                                                   ((equal (cadr i) 'value)
                                                   (setf largest 0) (setf multmatch nil)
                                                   (do ((theplist (cdr i nil))
                                                        ((or (null theplist) (equal largest 1)))
                                                        (setq j (car theplist))
                                                        (setq theplist (cdr theplist))
                                                        (cond
                                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                                           ; setq largest 1)
                                                          ((equal (cadr i) 'value)
                                                           (setf largest 0) (setf multmatch nil)
                                                           (do ((theplist (cdr i nil))
                                                                ((or (null theplist) (equal largest 1)))
                                                                (setq j (car theplist))
                                                                (setq theplist (cdr theplist))
                                                                (cond
                                                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                   ; setq largest 1)
                                                                   ((equal (cadr i) 'value)
                                                                   (setf largest 0) (setf multmatch nil)
                                                                   (do ((theplist (cdr i nil))
                                                                        ((or (null theplist) (equal largest 1)))
                                                                        (setq j (car theplist))
                                                                        (setq theplist (cdr theplist))
                                                                        (cond
                                                                          ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                           ; setq largest 1)
                                                                           ((equal (cadr i) 'value)
                                                                           (setf largest 0) (setf multmatch nil)
                                                                           (do ((theplist (cdr i nil))
                                                                                ((or (null theplist) (equal largest 1)))
                                                                                (setq j (car theplist))
                                                                                (setq theplist (cdr theplist))
                                                                                (cond
                                                                                  ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                   ; setq largest 1)
                                                                                   ((equal (cadr i) 'value)
                                                                                    (setf largest 0) (setf multmatch nil)
                                                                                    (do ((theplist (cdr i nil))
                                                                                         ((or (null theplist) (equal largest 1)))
                                                                                         (setq j (car theplist))
                                                                                         (setq theplist (cdr theplist))
                                                                                         (cond
                                                                                           ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                            ; setq largest 1)
                                                                                            ((equal (cadr i) 'value)
                                                                                            (setf largest 0) (setf multmatch nil)
                                                                                            (do ((theplist (cdr i nil))
                                                                                                 ((or (null theplist) (equal largest 1)))
                                                                                                 (setq j (car theplist))
                                                                                                 (setq theplist (cdr theplist))
                                                                                                 (cond
                                                                                                   ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                                    ; setq largest 1)
                                                                                                    ((equal (cadr i) 'value)
                                                                                                     (setf largest 0) (setf multmatch nil)
                                                                                                     (do ((theplist (cdr i nil))
                                                                                                         ((or (null theplist) (equal largest 1)))
                                                                                                         (setq j (car theplist))
                                                                                                         (setq theplist (cdr theplist))
                                                                                                         (cond
                                                                                                           ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                                            ; setq largest 1)
                                                                                                            ((equal (cadr i) 'value)
                                                                                                             (setf largest 0) (setf multmatch nil)
                                                                                                             (do ((theplist (cdr i nil))
                                                                                                                  ((or (null theplist) (equal largest 1)))
                                                                                                                  (setq j (car theplist))
                                                                                                                  (setq theplist (cdr theplist))
                                                                                                                  (cond
                                                                                                                    ((equal (car i) 'HEADER) ; dont make headers match for now
                                                                                                                     ; setq largest 1)
                                                     
```

```

;////////////////////////////////////////////////////////////////
;;; (setq percent (/ nummatches total))
;;; (percent))

;***** ExplodeCase *****
;;; This function takes a case and expands it into a hierarchical
;;; form: e.g. (computer (type (ibm (color ...)))) Rather than the
;;; MOP index-based form. These expanded cases can be VERY large
;;; in FANSYS.

;;; Then we'd get the differences back as:
;;; diff = (computer (type ibm)
;;;                   (size big)
;;;                   (color (red 1) (blue 2) (green 1)))
;;; Notice that we only get the ways that case1 differs from case2;
;;; not the ways case2 differs from case1. The function must be
;;; called with the parameters reversed to get these.

;;; This function is useful to find the differences between slots
;;; so that mops may be indexed based upon their differences.
;;; *****DO*****

(defun ExplodeCase (case)
  (let ((explodedcase '()))
    (for (l :in case)
      (do
        (cond
          ((name->mop (cadr l)) ;;; value=MOP
           (setq explodedcase (append explodedcase (list (mop-slots (name->mop (cadr l)))))))
          (t) ;;; value=atom
            (setq explodedcase (append explodedcase (list l)))))

        explodedcase)
      )))
  )

;***** Head-Functions *****
;;; Small functions to see if the head of an item matches that of a
;;; list (similar to assoc but works on lists) and another to get the
;;; actual head of the list.

(defun Head-Match (item somelist)
  (for (i :in somelist)
    (when (equal (car i) (car item))
      (first i)))

(defun Get-Head (item somelist)
  (for (i :in somelist)
    (when (and (equal (car i) (car item))
              (listp (cadr i)))
      (save i)))

;***** Merge-Diff *****
;;; Merges list of lists together...used in Find-Dif to clean up
;;; the final output.
;;; Cases and returns them.

(defun Merge-Diff (l)
  (let ((newlist nil))
    (for (i :in l)
      (do
        (setq newlist (append newlist (cadr i)))
        (setq newlist (my-remove-dup newlist)))))

;***** Find-Sim-Recursive *****
;;; Counterpart to Find-Dif-Recursive.
;;; This function finds all of the similarities between two exploded
;;; cases and returns them.

;;; For example, given:
;;; case1 = (computer (type ibm)
;;;                   (color (red 1) (blue 2) (green 3))
;;;                   (size big))
;;; case2 = (computer (type mac)
;;;                   (color (red 3) (blue 2) (green 1))
;;;                   (size big))
;;; Then we'd get the similarities back as:
;;; sim = (computer (color (blue 2))
;;;                   (size big))

;;; This is useful for computing norms and generalizations of MOPS.
;;; *****DO*****

;;; Then we'd get the similarities back as:
;;; sim = (computer (color (blue 2))
;;;                   (size big))

;;; This function finds all of the differences between two exploded
;;; cases and returns the differences.

;;; For example, given:
;;; (defun Find-Sim-Recursive (case1 case2)
;;;   (let ((matches nil) (value nil) (submatch nil))
;;;     (for (i :in case1)
;;;       (do
;;;         (setq matches (append matches (value i)))
;;;         (setq value (submatch i))
;;;         (setq submatch (value i))))))

;;; Then we'd get the similarities back as:
;;; sim = (computer (color (blue 2))
;;;                   (size big))

;;; This is useful for computing norms and generalizations of MOPS.
;;; *****DO*****

```

```

*****+
;; Elaborate-All
;; This function will add on all of the failure attributes for
;; FANSys when doing question answering if we want to elaborate
;; on everything. This is equivalent to resorting to a depth-first
;; brute-force search on everything since all indices can be
;; traversed.
;; Attributes of (X *) mean that X can be anything; any X index
;; will be traversed.
*****+
(for (j :in case2)
  :DO (cond
    ((equal (car j) (car i)) ;;; attributes match
     (setq value (cadr j))
     (cond
       ((equal (cadr j) value) ;;; exact match
        (setq matches (append matches (list j)))
        (and (listp (cadr j)) (listp value)) ;;; gotta recurse
        (setq submatch (Find-Sim-Recursive (cadr j) value))
        (cond
          (submatch (append matches (list (list (car j) submatch)))) ;;; matches
          ())) ;;; matches
        (My-Remove-Dupe matches)))
      ))) ;;; matches
  :Reformat
  ;;; After a MOP has been exploded using Explode-It, this just goes
  ;;; through the explosion and turns the output into a format
  ;;; which is consistent with the attribute/value'd indices structure.
  ;;; *****+
  (defun reformat (l)
    (let ((temp nil) (temp3 nil) (temp4 nil))
      (temp4 nil)
      (for (i :in l)
        :DO (cond
          ((atom i) (setq temp i))
          ((equal (length i) 2)
           (cond
             ((atom (cadr i)) (setq temp (append temp (list i))))
             (t (setq temp (append temp (list (car i) (reformat (cdr i)))))))) ;;; matches
           (t ;;; Leave abstraction out
            (setq temp2 (cddr i))
            ; Add mopname as "header" slot
            (setq temp2 (cons 'header (list (cadr i)))
                  (setq temp2 (cons temp2 (cddr i)))))

            ; Add mopname as slot attribute
            (setq temp2 (cddr i))
            (setq temp2 (cons (cadr i) (list temp2)))
            ; (setq temp2 (list temp2))
            (setq temp3 (reformat temp2))
            (setq temp4 (list (car i) temp3))
            (setq temp (append temp (list temp4))))))) ;;; matches
      )) ;;; HasRecursion
  ;;; Some of the FANSys definitions are recursive. This checks a
  ;;; definition to see if it refers to itself; if so, stop the
  ;;; recursion/expansion process as it would go forever. In particular,
  ;;; Software and Hardware systems are defined this way.
  ;;; *****+
  (defun HasRecursion (mop)
    ;;; Misc Functions
    ;;; *****+
    (defun slot-compare (a b)
      (string< (car a) (car b)))
    (defun car-member (item l)
      (for (i :in l)
        :WHEN (equal item (car i))
        :FIRST i)))

```


;;; hierarchy until we find one that does contain all the items and return
 ;;; that as the entry point.

;;;
 ;;; Basically, we are trying to find the most specific MOP which
 ;;; encapsulates all of the given information and start searching
 ;;; there rather than from some higher-level MOP such as M-root.
 ;;; It is important to note that this begins the search bottom-up, based
 ;;; upon the MOPs given as the REQUESTED information. So if we are
 ;;; asking for a failure-mode, start looking at failure mode and then
 ;;; start looking at abstractions and backpointers to determine the
 ;;; entry point.

;;;
 ;;; In the case of multiple desired MOPs, it's basically the same thing
 ;;; except we do an intersection of all the backpointers of the MOPs.
 ;;; Kind of messy due to the equivalence / abstraction constraints
 ;;; imposed by the parsing module.

;;;
 (defun Find-Entry-MOP (itemlist moplist)
 (let ((back nil))
 (and *verbose*
 (format nil "~>Searching bottom-up for Entry-MOP based on requested:~<"))
 (cond
 (equal (length moplist) 1)
 (cond
 ((Many-Slot-Abst itemlist) (car moplist))
 (t (setq back (GetBackPointer (car moplist)))
 (setq back (GetBackPointer (car moplist)))
 ; weed out IP's
 (setq back
 (for (i :in back)
 :WHEN (not (My-Member 'M-INDEX-PATTERN
 (mop-all-absts (name->mop 1))))
 :SAVE i))
 (for (i :in back)
 :WHEN (and (progn (and *verbose* (st (format nil "~S<" i))) 't)
 (Many-Slot-Abst itemlist 1))
 :FIRST i)))
 (t (setq back (GetBackPointer (car moplist)))
 (setq back
 (for (i :in back)
 :WHEN (not (My-Member 'M-INDEX-PATTERN
 (mop-all-absts (name->mop 1))))
 :SAVE i))
 (for (i :in (cdr moplist))
 :DO
 (setq back (Intersection back (GetBackPointer 1))))
 (setq back (my-remove-dups back))
 (cond (back
 (for (i :in back)
 :WHEN (and (progn (and *verbose* (st (format nil "~S<" i))) 't)
 (Many-Slot-Abst itemlist 1))
 :FIRST i))
 (t nil))))

;;;
 ;;; Find-Entry-MOP2

;;;
 ;;; In the course of implementation, some cases were not answerable by
 ;;; using the strategy outlined above.

;;;
 ;;; If the above Find-Entry MOP fails (we started looking from the requested
 ;;; MOPs and see if everything is

;;; indexed under it. This starts searching from the MOPs that are
 ;;; given, not from the requested information.

```
*****  

(defun Find-Entry-MOP2 (itemlist moplist)
  (let ((given nil) (found nil) (previous nil))
    (and *verbose*
         (st (format nil "~>Searching bottom-up for Entry-MOP based on given:~<" ))
         (format nil "~>Searching bottom-up for Entry-MOP based on given:~<" ))
         (setf given (append itemlist moplist))
         (for (i :in given)
              (WHEN
                (cond
                  ((lisp1) ;;; Not totally correct here. Should compare whole slot
                   (setf previous 1) ;;; previous MOP in abstraction hierarchy
                   (setf found 1) ;;; In Many-Slot-Abst
                   (for (j :in (mop-all-absts (name->mop (cadr i))))
                        :WHEN (cond
                            ((Many-Slot-Abst (my-remove i given) j)
                             (and *verbose* (st (format nil "~S<" j)))
                             (t)
                             (t (setf previous j) nil)))
                            (:FIRST previous)))
                  (t (setf previous 1) ;;; previous MOP in abstraction hierarchy
                   (setf found 1)
                   (for (j :in (mop-all-absts (name->mop 1)))
                        :WHEN (cond
                            ((Many-Slot-Abst (my-remove i given) j) 't)
                            (t (setf previous j) nil)))
                            (:FIRST (cond (listp found) (cadr found)
                                         (t found)))
                  ))))  

*****  

      ST  

*****  

      This is the trace-output function. It's defined here in case  

      the cmu client window isn't loaded (i.e, we are running in ACL).


*****  

      In this case, this function is used instead. If cmuload.lsp is loaded  

      then this function will be redefined to send output to the trace  

      client. ST = Send Trace.



*****  

      (defun st (obj) (princ obj) (terpri))


```

```

((name->mop 'M-PATTERN) (abstP (name->mop 'M-PATTERN) x))

*****  

;* FANSYS: A Computer Model of Text Comprehension and Question  

;* Answering for Failure Analysis  

;* File: memory.lsp  

;* Developed by: Sergio J. Alvarado  

;* Ronald K. Braun  

;* Kendrick J. Mock  

;* Artificial Intelligence Laboratory  

;* Computer Science Department  

;* University of California  

;* Davis, CA 95616  

;* Funds for the support of this study have been allocated by the  

;* NASA-Ames Research Center, Moffett Field, California, under  

;* Interchange No. NCA2-721.  

;*  

;* This file contains the lisp code necessary for the creation of FANSYS'  

;* Semantic Net Memory (the nodes of which are called MOPs for simplicity).  

;* This is essentially a modified version of microMOPs from Inside  

;* Case-Based Reasoning (ICBR) by Riesbeck and Schank; modifications  

;* are written in lower case.  

;* MicroMOPs' mops are reimplemented as structures instead of tables for  

;* efficiency and ease of access. This is the primary modification to these  

;* functions, and is not documented. See the original ICBR for  

;* comments on the original (uppercase) code. Any additional changes  

;* are documented in the code.  

(DEFUN MOPP (X)  

  (OR (NUMBERP X) (mop-p X)))  

(DEFUN INSTANCE-MOPP (X)  

  (AND (MOPP X)  

       (OR (NUMBERP X) (MOP-TYPE X) 'INSTANCE)))  

(DEFUN ABST-MOPP (X)  

  (AND (MOPP X) (MOP-TYPE X) 'MOP))  

(DEFUN ABSTP (abst spec)  

  ; the return value of abstP given a null argument and such have  

  ; altered repeatedly over the life of this parser; the following  

  ; choices seem to work well.  

  (cond  

    ((null spec) nil)  

    ((null abst) t)  

    ((eq (mop-name abst) (mop-name spec)) t)  

    ((member (mop-name abst) (mop-all-abssts spec)) t)  

    ; << added on 2/8/93 by r -- equivalence in abssts  

    ((member (mop-name abst) (mop-equivalent spec)) t)  

    ;>>  

    (t nil)))  

(DEFUN PATTERNP (X)  

  (insist patternP (or (null X) (mop-p X)))  

  (cond  

    ((name->mop 'M-PATTERN) (abstP (name->mop 'M-PATTERN) x))
    (DEFUN GROUPP (X)
      (insist groupP (mop-p X))
      (cond  

        ((name->mop 'M-GROUP) (abstP (name->mop 'M-GROUP) x))
        (t nil)))
      (DEFUN SLOT-ROLE (SLOT) (CAR SLOT))
      (DEFUN SLOT-FILLER (SLOT) (CAR (CDR SLOT)))
      (DEFUN MAKE-SLOT (ROLE mopname) (LIST ROLE mopname))
      (DEFUN ROLE-SLOT (ROLE mopname)
        (let ((mop (name->mop mopname)))
          (INSIST ROLE-SLOT (ROLE mopname)))
        (OR (MOPP mop) (LISTP mopname)))
        (ASSOC ROLE
          (COND
            ((MOPP mop) (MOP-SLOTS mop))
            ((T mopname))))))
      (DEFUN ROLE-FILLER (ROLE X)
        (insist role-filler (not (mopP X)))
        (SLOT-FILLER (ROLE-SLOT ROLE X)))
      (DEFUN ADD-ROLE-FILLER (ROLE mopname FILLER)
        (let ((mop (name->mop mopname)))
          (INSIST ADD-ROLE-FILLER (ROLE mopname FILLER))
          (MOPP MOP) (NULL (ROLE-FILLER ROLE mopname)))
        (and *debug* (format T "~-S~-S~-S~" mopname ROLE FILLER))
        (SETF (MOP-SLOTS MOP)
              (CONS (MAKE-SLOT ROLE FILLER)
                    (CONS (MOP-SLOTS MOP)
                          (MOP-SLOTS MOP))))))
      FILLER)))
      (DEFUN LINK-ABST (SPEC ABST)
        (INSIST LINK-ABST (ABST-MOPP ABST)) (MOPP SPEC)
        (COND ((NOT (ABSTP ABST SPEC))
               (SETF (MOP-ABSTS SPEC)
                     (CONS (mop-name ABST) (MOP-ABSTS SPEC))))
               (SETF (MOP-SPECS ABST)
                     (remove-duplicates (CONS (mop-name SPEC)
                                              (MOP-SPECS ABST))))))
               (REDO-ALL-ABSTS SPEC)))
      (DEFUN UNLINK-ABST (SPEC ABST)
        (insist unlink-abst (mopP spec) (mopP abst))
        (COND ((ABSTP ABST SPEC)
               (SETF (MOP-ABSTS SPEC)
                     (REMOVE- (mop-name ABST) (MOP-ABSTS SPEC))))
               (SETF (MOP-SPECS ABST)
                     (REMOVE- (mop-name SPEC) (MOP-SPECS ABST))))))
        (REDO-ALL-ABSTS SPEC)))
      (DEFUN REDO-ALL-ABSTS (MOP)
        (insist redo-all-abssts (mopP MOP))
        (SETF (MOP-ALL-ABSTS MOP) (CALC-ALL-ABSTS MOP))
        (FOR (SPEC :IN (MOP-SPECS MOP))
             :DO (REDO-ALL-ABSTS (name->mop SPEC)))))

(DEFUN CALC-ALL-ABSTS (MOP)

```

```

(for (slot :in (get-all-slots mop))
  (list calc-all-abssts (moppp mop))
  (REMOVE-DUPLICATES
    (CONS (mop-name MOP) (FOR (ABST :IN (MOP-ABSTS MOP))
      :SPLICER (MOP-ALL-ABSTS (name->mop ABST))))))
  :save (let* ((slotmop (name->mop (cadr slot)))
    (vars (cdr (lassoc slot (mop-vars mop)))))

    ;<<< added 10/25/92 by r ***var bindings***

    ; The following mess of functions implement a variable bindings
    ; package. It is useful to specify variables within the
    ; representation, to constrain slot fillers across a representation to
    ; be the same, and to automatically generate inferences by
    ; instantiating variables over the entire representation. This
    ; package is implemented using tricky list pointer manipulations, such
    ; as nconc and replace!, which makes it efficient and fast but
    ; non-comprehensible.

    ; Function var-bindings expands out the representation, binding
    ; variables across all levels of the representation, creates a mop for
    ; the new (fleshed out) expansion, and then indexes that mop in Ken's
    ; hierarchy (assuming we want it to do so, as specified by
    ; *generalize*).

    (defun bind-variables (mop)
      (let ((rep nil)
            (bindings (copy-list '())))
        (bindings (copy-list 'nil)))
        ; expand out the node given existing slots and the var bindings
        ; specified by the representation
        (setf rep (expand-node mop bindings nil))
        ; get rid of everything in the expanded representation that is not
        ; part of an instance mop (i.e. that is not actually bound)
        (setf rep (mop-to-instance (trim-rep rep)))
        ; create the new mop for this expansion
        (setf newmop (new-mop (mop-name mop)
          (mop-absts mop)
          (mop-type mop)
          (forms->slots rep)))
        ; since new-mop sidesteps the generalization in slots->mop, call
        ; generalize explicitly
        (when (let ()
          (generalize*
            (integrated-add-instance (mop-slots newmop) (mop-name newmop)
              (car (mop-absts newmop))))))

        newmop))

    ; Function trim-rep gets rid of the parts of the expanded
    ; representation that are not bound to instance mops.

    (defun trim-rep (rep)
      (let ((trim nil))
        (for (slot :in rep)
          :when (let ()
            (setf trim (trim-rep (cdddr slot) 'mop)
              (not (and (equal (caddr slot) (caddr slot))
                (null trim)))))

            :save (let ((newrep (list (car slot) (caddr slot) 'mop)
              (if trim
                (setf newrep (append newrep trim))
                newrep)))))

        newrep))

    ; Function expand-node takes an existing mop, with some slot fillers,
    ; expands out that mop explicitly, then applies variable bindings to
    ; expand out the rest of the representation.

    (defun expand-node (mop bindings visited)
      ; expand out each slot of the mop to construct the full expansion

```

```

; actual binding should be -- now set all variables to
; that representation.  Ware the ncons and rplacs!
(for (var :in vars)
  (do (cond
        ((and (assoc var bindings)
              (abstp (name->mop) (caddr (assoc var b
                (name->mop) (caddr filler-ptr)))
              (not (eq (cadr (assoc var bindings))
                        (cdr filler-ptr)))
              (equal (car (cddadr (assoc var binding
                ,MOP)))
        (cond
          ((equal (caddr filler-ptr) 'instance)
           (rplaca (caddr (assoc var bindings))
                    (caddr filler-ptr)))
          (rplacd (caddr (assoc var bindings))
                  (cdddr filler-ptr)))
          (t
            (rplacd (cadr (assoc var bindings))
                    (cdr filler-ptr))
            (setf (cddadr (assoc var bindings))
                  (cdr (caddr (assoc var bindings)))
                  (assoc var bindings)))
            (t
              (inccong bindings
                        (list (list var (cdr
                            (filler-ptr)))))))
        ; filler-ptr contains the expanded subrep -- return i
        filler-ptr)))
      ; This function converts the expanded representation into one that can
      ; be passed to slots->mop by adding instance tags to slots.
      (defun mop-to-instance (rep)
        (for (elem :in rep)
          :save (cond
                  ((equal elem 'MOP) 'INSTANCE)
                  ((atom elem) elem)
                  (t (mop-to-instance elem)))))

      ; The next few functions allow for list manipulations using either
      ; structural or list pointer access.

      (defun lassoc (elem 1)
        (for (k :in 1)
          :when (equal (car k) elem)
          :first k))

      (defun lassoc (elem 1)
        (for (k :in 1)
          :when (eq (car k) elem)
          :first k))

      (defun lmember (elem 1)
        (for (k :in 1)
          :when (equal k elem)
          :first t))

      (defun rremove (elem 1)
        (remove-if #'(lambda (x) (equal elem x)) 1))

      ; Get-all-slots collects together all of the slots for a mop,
      ; traversing up the abstraction hierarchy to fetch abstract slots that
      ; weren't bound locally.  This function should get all of the slot

```



```

; << added 10/5/92 r **backpointers**
; unlink backpointers
; (and (not ip-flag)
;      (for (slot :in (mop-slots mop))
;           :do (let ((backmop (name->mop (cadr slot)))
;                    (and (null backmop)
;                         (and *verbose*
;                              (st (format nil
;                                         "~~Error: used undefined ~s in ~s."
;                                         (cadr slot) name)))))))
;                (setf (mop-back backmop)
;                      (remove-if #'(lambda (x)
;                                     (equal (list (car slot) name) x))
;                                 (mop-back backmop))))))
;           ;>>
;           (setf *mops* (delete-key *mops* name)))
;           (mop)))
;      (DEFUN INHERIT-FILLER (ROLE mopname)
;        (insist inherit-filler (not (mopp mopname)))
;        (FOR (ABST :IN (MOP-ALL-ABSTS (name->mop mopname)))
;             :FIRST (ROLE-FILLER ROLE ABST)))
;          (DEFUN GET-FILLER (ROLE mopname)
;            (insist get-filler (not (moppp mopname)))
;            (OR (ROLE-FILLER ROLE mopname)
;                (LET ((FTILLER (INHERIT-FILLER ROLE mopname)))
;                  (AND FTILLER
;                       (OR (AND (INSTANCE-MOPP (name->mop FILLER)) FILLER)
;                           (AND (ABSTP (name->mop 'M-FUNCTION)) (name->mop FILLER))
;                               (LET ((FN (GET-FILLER 'CALC-FN FILLER)))
;                                   (AND FN
;                                        (LET ((NEW-FILLER (FUNCALL FN FILLER mopname)))
;                                            (AND NEW-FILLER
;                                                 (ADD-ROLE-FILLER ROLE mopname
;                                                       NEW-FILLER))))))))
;                           (DEFUN PATH-FILLER (PATH mopname)
;                             (insist path-filler (not (moppp mopname)))
;                             (AND (FOR (ROLE :IN PATH)
;                                       :ALWAYS (SETF mopname (GET-FILLER ROLE mopname)))
;                                  MOP)))
;                               (DEFUN SLOTS-ABSTP (MOP SLOTS)
;                                 (insist slots-abstp (and (moppp mop) (moppp slots)))
;                                 (NOT (NULL (MOP-SLOTS MOP)))
;                                   (NOT (NULL (MOP-SLOTS MOP)))
;                                     ; may want to insure that mops are related by abst hierarchy, to
;                                     ; prevent comparing apples and oranges
;                                     (or (null (mop-slots slots))
;                                         (for (slot :in (mop-slots mop))
;                                              (when (assoc (car slot) (mop-slots slots))
;                                                (first t)))
;                                              (FOR (SLOT :IN (MOP-SLOTS MOP))
;                                                :when (assoc (car slot) (mop-slots slots))
;                                                :ALWAYS (SATISFYEDP (SLOT-FILLER SLOT)
;                                                       (GET-FILLER (SLOT-ROLE SLOT)
;                                                       (mop-name slots)))
;                                                       (mop-name slots)))))))
;                               (DEFUN SATISFYEDP (CONSTRAINT-FILLER SLOTS)
;                                 (insist satisfyedp (not (moppp slots)))
;                                 (COND ((NULL CONSTRAINT)
;                                         (PATTERNP (name->mop CONSTRAINT)))
;
```



```

DEFUN GROUP-SIZE (X)
  (AND (GROUPP X) (LENGTH (MOP-SLOTS X)))
  (DEFUN GROUP->LIST (GROUP)
    (AND GROUP
         (INSIST GROUP->LIST (GROUPP GROUP))
         (FOR (INDEX :IN (MAKE-M-M-1 (GROUP-SIZE GROUP)))
              :IFILTER (ROLE-FILLER INDEX (mop-name GROUP)))))

(DEFUN LIST->GROUP (L)
  (COND ((NULL L) 'I-M-EMPTY-GROUP)
        (T (SLOTS->MOP
          (FOR (X :IN L)
            (T :IN (MAKE-M-M-1 (LENGTH L)))
            (SAVE (MAKE-SLOT T X))
            ' (M-GROUP)
            T)))))

(DEFUN MAKE-M-M-N (M N)
  (INSIST MAKE-M-M-N (INTEGERP M) (INTEGERP N))
  (COND ((EQL M N) (LIST N))
        ((< M N) (CONS M (MAKE-M-M-N (+ M 1) N)))
        (T (CONS M (MAKE-M-M-N (- M 1) N))))))

(DEFUN DAH (mopname)
  (PPRINT (TREE->LIST (name->mop mopname) #'SPEC->LIST NIL)))

(DEFUN DPH (mopname)
  (PPRINT (TREE->LIST (name->mop mopname) #'SLOTS->FORMS NIL)))

(defun explode-it (mopname)
  (tree->list (name->mop mopname) #'slots->forms nil))

(DEFUN SPECS->LIST (MOP VISITED)
  (FOR (SPEC :IN (MOP-SPECS MOP))
       :SAVE (TREE->LIST (name->mop SPEC) #'SPEC->LIST VISITED)))

(DEFUN SLOTS->FORMS (MOP VISITED)
  (FOR (SLOT :IN (MOP-SLOTS MOP))
       :SAVE (CONS (SLOT-ROLE SLOT)
                  (MOP->FORM (SLOT-FILLER SLOT)
                               VISITED)))))

(DEFUN MOP->FORM (mopname VISITED)
  (cond
    ((listp mopname) (List mopname)
      (t (TREE->LIST (name->mop mopname) #'SLOTS->FORMS VISITED)))))

(DEFUN TREE->LIST (MOP FN VISITED)
  (COND ((MEMBER MOP VISITED) (LIST (mop-name MOP)))
        (T (SETF VISITED (CONS MOP VISITED))
           ' (, (mop-name MOP) , (FUNCALL FN MOP VISITED)))))

(DEFUN CONSTRAINT-FN (CONSTRAINT FILLER SLOTS) T)
  (DEFUN GET-SIBLING (PATTERN mopname) (not (mopp mopname)))
  (DEFUN GET-SIBLING (and (not (mopp pattern)) (not (mopp mopname)))
```



```

(store 1)
(geno nil)
(nogeno nil)
(optno nil)
(optlist nil)
(terminatoro nil)
(terminatoro 'm-hard-punctuation)
; if a syntax category is specified, then use it;
; this code segment implements backward compatibility
(cond
  (and (atom (car phrase))
    (name->mop (car phrase)))
  (abstp (name->mop 'm-syntax-category)
    (name->mop (car phrase))))
  (setf syntax (car phrase))
  (setf phrase (cdr phrase)))
  (t
    (and *verbose*
      (st (format nil
        "Warning: inferring default syntax for phrase ~s."
        phrase)))))
; find storeno and terminatorno and optno
(setf genno (position 'gen phrase))
(setf nogeno (position 'nogen phrase))
(setf storeno (position 'store phrase))
(setf terminatorno (position 'terminator phrase))
(setf optno (position 'opt phrase))
; decompose command elements out of the phrase
(cond
  (storeno
    (setf store (nth (1+ storeno) phrase))
    (setf (elt phrase storeno) nil)))
  (cond
    (genno
      (setf (elt phrase genno) nil)))
    (cond
      (nogeno
        (setf (elt phrase nogeno) nil)))
    (cond
      (terminatorno
        (setf terminator (nth (1+ terminatorno) phrase))
        (setf (elt phrase terminatorno) nil)))
      (cond
        (optno
          (setf optlist (nth (1+ optno) phrase))
          (setf (elt phrase optno) nil))
        (setf (elt phrase (1+ optno)) nil)))
      (setf phrase (remove-if #'(lambda (x) (equal nil x)) phrase))
      (deine-phrase 'mopname syntax phrase terminator store optlist
        (genno nogeno)))
    (cond
      ; Function define-phrase does the work of creating the phrase mop, storing
      ; the phrase as appropriate for generation, etc.
      ; In define-phrase:
      ; -- terminator: replaced with the actual spec of m-terminator that
      ; will do the termination
      (defun define-phrase (mopname syntax phrase terminator store optlist genno
        nogeno)
        (let ((ip (gentemp (format nil "-s." 'M-IP)))
          (oldterm terminator)

```

```
(cons ip (mop-ip-refs (name->mop mophame))))  
(and (null genno)  
     (setf (mop-ips storemop) (cons ip (mop-ips storemop)))))
```

```

; 5. If this node was predicted, then the target node of the prediction is
; activated. This for all such predictions.
;
; 6. All of the abstractions of this node are activated, and are passed the
; node's AM.
;
; Ideally, steps 3-6 should be order independent. This implementation is
; less than ideal, however. 'Ware changing the order!
;

(* File: parser.lsp
  * FANSYS: A Computer Model of Text Comprehension and Question
  * Answering for Failure Analysis
  * File: parser.lsp

  * Developed by: Sergio J. Alvarado
  *                 Ronald K. Braun
  *                 Kendrick J. Mock
  *
  * Artificial Intelligence Laboratory
  * Computer Science Department
  * University of California
  * Davis, CA 95616
  *

  * Funds for the support of this study have been allocated by the
  * NASA-Ames Research Center, Moffett Field, California, under
  * Interchange No. NCA-721.
  *****
  * These are the functions necessary for the parsing of text.

  ; The function parse gets a word at a time from the input text, and activates
  ; the lexical node for that word. If the word is not recognized, a new node
  ; is created for it; that node is indexed under m-unknown word. The mop
  ; m-done-with-word is activated after reading each word, so that various
  ; post-word activities (such as incrementing the word counter, killing ips,
  ; etc.) can be done *after* all normal word processing is finished.

  (defun parse (text)
    (format t "~>~%Parsing ~s." text)
    (for (word :in text)
      :do (and *verbose* (st (format nil "~>~%Reading ~s" word)))
      (cond
        ; unknown words are skipped to reduce brittleness of ips
        ((abstp (name->mop 'm-unknown-word) (name->mop word))
         (and *verbose*
              (st (format nil
                        "~~~~~ Skipping ~s: it is indexed as an unknown-word."
                        word))))
        ; normal word -- activate it
        ((name->mop word)
         (activate word t nil t)
         (activate 'm-done-with-word t nil t)
         (activate 'm-activated t nil t)
         (word-not-recognized -- execute appropriate failure strategy;
           ; may want to pull this out of parse eventually
           (t
             (activate 'f-unknown-word (list (word word) nil t)))))

  ; Function activate implements the referencing of a semantic node. When a
  ; node is activated, six steps occur:
  ; 1. The mop is specialized to an appropriate instance, if there are any
  ; specific slots passed to activate.
  ; 2. An activation marker is created for the node. If the node is activated
  ; with an AM passed to it, it gets a copy of that AM. Otherwise, a new
  ; AM is created.
  ; 3. The assoc-fn associated with that node is executed, if there is one.
  ; This function encodes the structure-specific (semantic) content of that
  ; node: i.e. how to process it.
  ; 4. All of the ips which have as their first element this node are
  ; activated.

  ; This function activates (mopname slots marker new)
  ; (insist activate (and (atom mopname) (or (null marker) (am-p marker))) )
  ; specialize mop into an instance that best characterizes it (this means
  ; creating a new mop if this is a brand new occurrence of these slots)
  ; (and (list slots)
  ;      (setf mopname (specialize-mop mopname slots)))
  ;      (and *debug* (format t "~-Activating ~s" mopname))
  ;      (<< mode checking added 1/20/93 by r
  ;      ; modes are stopgap measures designed to allow for contextual control;
  ;      ; don't do any activation if not in a constrained mode
  ;      (cond
  ;        ; mode was an unfortunate slot designator, since cases naturally
  ;        ; have modes -- therefore, ignore cases
  ;        ; (not (member 'm-case (mop-all-absts (name->mop mopname))))
  ;        ; do nothing if this is the wrong mode
  ;        ; (not (equal (inherit-filler 'mode mopname) *mode*)))
  ;        (t
  ;          ;>>> (let ((mop (name->mop mopname)))
  ;                  (cond
  ;                    ; for everything but processing mops and the root, call Ken's
  ;                    ; indexing stuff if doing normal parsing
  ;                    ; (member 'm-processing-mop (mop-all-absts mop))
  ;                    ; (equal (mop-name mop) 'm-root)
  ;                    ; ((instance-mopp mop)
  ;                    ;   (setf *generalize* t)
  ;                    ;   (setf mop (bind-variables mop)))
  ;                    ;   (setf mop (bind-variables mop))
  ;                    ;   (create-am mop mopname marker new)
  ;                    ;   (execute-assoc-fn mopname)
  ;                    ;   (activate-lps mop)
  ;                    ;   (respond-to-preds mop mopname)
  ;                    ;   (activate-absts mop mopname)
  ;                    ;   (mopname)))
  ;                    ; This function refines the mop, given a set of slots. An instance with
  ;                    ; those slots is found, if one exists; otherwise, a new instance is created.
  ;                    ; Note that an instance mop is always returned. The workhorse of this
  ;                    ; function is slots->mop -- this should be changed soon to encompass prior
  ;                    ; referents and various failure strategies.
  ;                    (defun specialize-mop (mopname slots)
  ;                      (let ((oldmopname mopname)
  ;                            (and *verbose* (st (format nil "~-f_ Recognizing: ~s" mopname)))
  ;                                ; slots->mop does the actual work of specializing
  ;                                ; (setf mopname (slot-s->mop (cons 'instance slots) (list mopname) t))
  ;                                ; (or (equal oldmopname mopname)
  ;                                ;     (and *verbose* (st (format nil "~-f_ Specializing: ~s" mopname))))))

  ; This function creates a new activation marker for the mop. This AM is
  ; added to the beginning of the mops am's list. If a marker is passed in,
  ; then that marker is used; otherwise, a new AM is created, with this mop
  ; as its parent. If a new marker is created, the start and finish slots
  ; from any prior marker is copied, to facilitate implementation of ip
  ; adjacency requirements. Parmaeter notes:

```

```

;
; -- new: flag; t = Create new marker, nil = use given marker
;
; -- marker: marker to be used for this node, or to be cannibalized
;   for start/finish properties

(defun create-am (mop mopname marker new)
  (let ((newmarker (cond (new
                                (cond (marker (make-am :owners (list mopname)
                                              :parent (mopname
                                              :start (am-start marker)
                                              :finish (am-finish marker)))
                                (t (make-am :owners (list mopname)
                                              :parent (mopname
                                              :start (get-counter)
                                              :finish (get-counter))))))
                               (t marker)))
                               ; add it to the list of owners for this AM, if it isn't already there
                               (or (member mopname (am-owners newmarker))
                                   (setf (am-owners newmarker)
                                         (cons mopname (am-owners newmarker)))))
                               ; add it to the front of the ams list for the mop
                               (setf (mop-ams mop) (acons newmarker (remove newmarker (mop-ams mop)))))
                               (and *debug* (format t "~>Created activation marker: ~s"
                                         (car (mop-ams mop)))))

; This function executes the assoc-fn for this node, if one exists.

  (defun execute-assoc-fn (mopname)
    (let ((assoc-ins (mop-assoc-fns (name->mop mopname))))
      (and *debug*
           (format t "~>Executing assoc-fns of ~s: ~s" mopname assoc-fns)
           (for (assoc-fn :in assoc-fns)
             :do (funcall assoc-fn mopname)))))

; This function activates all of the index patterns which have as their first
; element this node.

  (defun activate-ips (mop)
    (for (lip :in (mop-ips mop))
      :do (activate lip
                    (list 'index 1 'termflag nil)
                    (list 'current 'am-start (car (mop-ams mop)))
                    (list 'start (car (mop-ams mop))))))
      (car (mop-ams mop)) nil))

; Function respond-to-preds handles any pending predictions.
; If the mop passed in was predicted (i.e. there are PMs on its PMS list),
; then the targets of the predictions are activated, and the PMs are deleted.
; It is important that preds are deleted before their targets are activated;
; I don't remember why, but I think it may be that failure to do so might
; induce an endless recursion, if this same mop is reactivated, and reuses
; the prediction ad infinitum. Note that if the target of the mop is no
; longer existant, the prediction is simply ignored (crude pm clean-up).

  (defun respond-to-preds (mop mopname)
    (cond
      ((mop-pms mop)
       (for (pred :in (mop-pms mop))
         :do (and *debug* (format t "~>Fulfilled prediction: ~s" pred))
             (setf (mop-pms mop) (remove pred (mop-pms mop)) :count 1)
             (and (mop (name->mop (pm-parent pred)))
                  (activate (pm-parent pred) t (car (mop-ams mop)) nil)))))

; Function activate-abscts activates up the abstraction hierarchy. All
; abstractions of this node are activated. This node's AM is passed


```

```

(defmop m-context (m-processing-mop)
  (context m-root)
  ; punctuation
  (defmop m-punctuation (m-lexicon))
  ; hard punctuation causes the termination of some index-patterns (those with
  ; m-hard-punctuation as their terminators), while soft punctuation does not.
  (defmop m-hard-punctuation (m-punctuation))
  (defmop *period* (m-hard-punctuation) instance)
  (defmop *exclamation* (m-hard-punctuation) instance)
  (defmop *eoe* (m-hard-punctuation) instance)
  (defmop *eoc* (m-hard-punctuation) instance)
  (defmop m-soft-punctuation (m-punctuation))
  (defmop *comma* (m-soft-punctuation) instance)
  ; The mop for m-index-pattern has the following slots:
  ;   -- phrase: the actual phrase that constitutes the IP
  ;   -- target: the mop to be activated upon recognition of all elements
  ;   -- index: a pointer to the element currently expected
  ; Additionally, specs of m-index-pattern have these slots:
  ;   -- terminal: flag, set if the terminator node has been predicted for
  ;     this IP
  ;   -- current: the counter for the required start of the next element to
  ;     be bound; used to enforce adjacency requirements
  ;   -- start: the counter at the time the IP was created
  ; -----
  (defmop m-index-pattern (m-processing-mop) (phrase nil) (target nil)
    (index nil))

  (add-assoc-fn m-index-pattern ip-fn)
  ; m-unknown-word indexes unknown words that crop up during a parse
  (defmop m-unknown-word (m-lexicon))
  ; this MOP records the fact that processing has finished for a given word;
  ; it is sometimes useful to let all other higher level processing finish
  ; before taking some action (like deleting an index pattern); a counter
  ; is also incremented after each lexical element has been read
  (defmop m-done-with-word (m-processing-mop) (count nil))
  (replace-slot 'm-done-with-word 'count 1)
  (add-assoc-in m-done-with-word incr-count-fn)

  ; Function incr-count-fn increments the word counter. This counter is
  ; sometimes used to test for adjacency of words and such.
  (defun incr-count-fn (mopname)
    (let* ((mop (name->mop mopname))
           (count (cadr (assoc 'count (mop-slots mop)))))
           (replace-slot mopname 'count (1+ count)))
      (replace-slot mopname 'count (1+ count)))

  ; these MOPs implement the simple syntax categories required for generation;
  ; obviously, a more systematic treatment of syntax is in order.
  (defmop m-syntax-category (m-processing-mop))

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: parser.mops

Developed by: Sergio J. Alvarado
               Ronald K. Braun
               Kenrick J. Mock

Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616

Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

These memory structures encode lexical knowledge.

All mops (such as index patterns, lexical entries, special processing
mops, etc.) that are used in flow of control processing should be
indexed as m-processing-mops.

(defmop m-processing-mop (m-root)
  ; -----
  ; The following mops are hold-overs from the Inside Case-Based
  ; Reasoning definitions by Riesbeck and Schank.
  ; These may be deleted at some point.

(DEFMOP M-GROUP (M-PROCESSING-MOP))
(DEFMOP M-EMPTY-GROUP (M-EMPTY-GROUP) INSTANCE)
(DEFMOP 1-M-EMPTY-GROUP (M-EMPTY-GROUP) INSTANCE)

(DEFMOP M-FUNCTION (M-PROCESSING-MOP))
(DEFMOP CONSTRAINT-FN (M-FUNCTION))
(DEFMOP CONSTRANT-FN (M-PROCESSING-MOP) (ABST-FN CONSTRAINT-FN))

(DEFMOP M-PATTERN (M-PROCESSING-MOP) (ABST-FN CONSTRAINT-FN))

(DEFMOP GET-SIBLING (M-FUNCTION))
(DEFMOP M-ROLE (M-PROCESSING-MOP))
(DEFMOP NOT-CONSTRAINT (CONSTRAINT-FN))
(DEFMOP M-NOT (M-PATTERN) (ABST-FN NOT-CONSTRAINT))

(DEFMOP M-FAILED-SOLUTION (M-PROCESSING-MOP))
  ; -----
  ; words, punctuation, etc. are stored in the lexicon
  (defmop m-lexicon (m-processing-mop))
  ; context is used to keep track of same case processing contexts during a
  ; parse.

```

```
(defmop s-np (m-syntax-category))
(defmop s-clause (m-syntax-category))
(defmop s-vp (m-syntax-category))
(defmop s-pp (m-syntax-category))
```

```

(remove-mop ipname))
  (try-to-advance-ip ipname mop)))))

Function fill-pre-store-slots returns t if all of the pre-store nodes could
be filled, nil otherwise.

(defun fill-pre-store-slots (ipname)
  (let ((pre-slots (make-mn (l- (inherit-filler 'store ipname)) 1))
        (optionals (inherit-filler 'optionals ipname))
        (all-filled t))
    ; assume all pre-store slots can be filled; if we get to one that can't
    ; be filled, invalidate that assumption
    (for (slot :in pre-slots)
         :when (and (not (fill-slot ipname slot))
                    (not (member slot optionals)))
         :do (setf all-filled nil)
            (all-filled)))

  Function fill-slot tries to fill in the given slot. If successful, t is
  returned, otherwise nil.

(defun fill-slot (ipname pos)
  (let* ((filler (inherit-filler pos ipname))
        ; elem is the target slot filler
        (elem (cond
                  ((atom filler) filler)
                  ((caddr filler)))
        ; see if an activation marker exists for the target slot filler
        ; which fulfills the adjacency requirements of the phrase
        (marker (find-marker elem (inherit-filler 'start ipname))))
    (cond
      ; if no marker can be found, then the slot filler was not recognized
      ; from the phrase; return nil
      ((null marker) nil)
      ; if the marker was found, and this is a word, then just update the
      ; ip start slot to be the start pos of this word
      ; (atom filler)
      ((replace-slot ipname 'start (am-start marker)))
      ; otherwise, fill the slot, and update start as above
      (t
       (replace-slot ipname pos (list (car filler) (cdr filler)
                                      (am-parent marker)))
       (replace-slot ipname 'start (am-start marker))))))

  Function find-marker scans the markers for the target slot and returns the
  first one which fulfills the phrase adjacency requirement. This requirement
  specifies that the slot has to end just prior to the beginning of the current
  start slot of the ip.

(defun find-marker (elem start)
  (for (am :in (mop-ams (name->mop elem)))
       :first (equal (am-finish am) (l- start))
       :do am))

  Function try-to-advance-ip checks to see if the ip is advanceable (i.e.
  ; the current am for the ip matches the next element (if a word) or is an
  ; abstraction of the slot filler for the next element (if not a word)).
  ; start slot of the ip.

(defun try-to-advance-ip (ipname mop)
  (let* ((lmarker (car (mop-ams mop)))
        (curpos (find-curpos lmarker ipname mop)))
    (cond (curpos
           (unless (try-to-advance-ip curpos)
             ; if this ip just got activated (as determined by termflag), then

```

```

; activate its terminator node
(cond
  (null (role-filler 'termflag ipname))
  (replace-slot ipname 'termflag t)
  (activate (inherit-filler 'terminator ipname) nil
    (make-am :parent ipname :owners (list ipname)) nil))
  (cond
    ((advanceablep (inherit-filler curpos ipname) ipmarker ipname curpos)
     (advance-ip ipname mop ipmarker curpos
       (inherit-filler 'length ipname)))
    ; if not advanceable, remove the index pattern (or at least do
    ; *something* with the IP
    (t
      (and *debug* (format t "~-tNote: advance-ip failed -- repredicting ~s." ipname)
        (format t "~-tNote: advance-ip failed -- repredicting ~s." ipname)
        ; may want to delete the IP here, instead of repredicting it,
        ; once the kill order is better specified
        ; predict-next-element (inherit-filler curpos ipname) ipname)))
      (remove-mop ipname))
    ; Since some elements of a phrase may be optional, find-curpos determines
    ; which position in the phrasal pattern had its prediction fulfilled. It
    ; simply steps through the slots from index on until it finds a match for
    ; the activated node (as specified through ipmarker).
    (defun find-curpos (ipmarker ipname mop)
      (let ((curpos (inherit-filler 'index ipname)))
        (for (i :in (make-mn curpos (inherit-filler 'length ipname)))
          :first (let ((filler (inherit-filler i ipname)))
            (cond
              ((atom filler)
                (and (abstp (name->mop filler)
                  (name->mop (am-parent ipmarker)))
                  1))
              (t
                (and (abstp (name->mop (caddr filler))
                  (name->mop (am-parent ipmarker)))
                  i)))))))
      (t
        (and (abstp (name->mop (caddr filler))
          (name->mop (am-parent ipmarker)))
            1)))
    (defun advanceablep (curelem ipname curpos)
      (cond
        ; if this is not the storage element in the pattern, make sure adjacency
        ; requirements of the IP are met
        ; (and (not (equal curpos (inherit-filler 'store ipname)))
        ;       (let* ((newpos (inherit-filler 'fill-next ipname))
        ;             (newelem (inherit-filler newpos ipname)))
        ;           (not (scan-markers-for-fit newelem ipmarker ipname))))
        ;       nil)
        (atom curelem)
        (equal (am-parent ipmarker) curelem)
        (t
          (abstp (name->mop (caddr curelem))
            (name->mop (am-parent ipmarker))))))

; Advanceablep checks to see if the next element matches the last activated
; node or is an abstraction of the last activated node. This determines
; whether the pattern can be advanced.
(defun advanceablep (curelem ipmarker ipname curpos)
  (cond
    ; if this is not the storage element in the pattern, make sure adjacency
    ; requirements of the IP are met
    ; (and (not (equal curpos (inherit-filler 'store ipname)))
    ;       (let* ((newpos (inherit-filler 'fill-next ipname))
    ;             (newelem (inherit-filler newpos ipname)))
    ;           (not (scan-markers-for-fit newelem ipmarker ipname))))
    ;       nil)
    (atom curelem)
    (equal (am-parent ipmarker) curelem)
    (t
      (abstp (name->mop (caddr curelem))
        (name->mop (am-parent ipmarker))))))

; Function advance-ip advances the IP (amazing!). When a pattern is advanced,
; the information (slot filler) for the last element is supplied by appending
; it to that slot in the IP. So, an element of the IP is not truly recognized
; until the next element is referenced. This prevents premature binding of a
; slot, by allowing subsequent (higher-order) processing to occur.

; If there is still more of the pattern left, the next element in the IP is
; predicted. Otherwise, since the pattern is completed, the target node is
; activated and the IP destroyed.
(defun advance-ip (ipname mop ipmarker curpos length)
  ; record the slot filler for the previous element
  (fill-last-role ipname (inherit-filler 'fill-next ipname) ipmarker t)
  (replace-slot ipname 'fill-next curpos)
  ; Increment index to next element
  (cond
    ((equal curpos length)
     ; the pattern is finished -- recognize it
     ; first, record the slot filler for the last element
     ; (fill-last-role ipname curpos ipmarker nil)
     (fill-last-role ipname curpos ipmarker nil)
     (and *verbose* (st (format nil "~-s = ~s referenced"
       (print-pattern (1+ curpos) length ipname)
       (inherit-filler 'target ipname))))
      ; next, activate the target function
      (let ((target (inherit-filler 'target ipname)))
        (activate target
          (get-ip-slots ipname)
          (make-am :start (role-filler 'start ipname)
            :finish (get-counter)
            nil)
          t)
        ; finally, remove the IP
        (remove-mop ipname)
        nil)
      (st (format nil "~-s = ~s" (print-pattern (1+ curpos) length ipname)
        (inherit-filler 'target ipname)))))
    ; there is more of the pattern -- save new pattern and index
    (replace-slot ipname 'index (1+ curpos))
    ; predict next element, unless current element was optional
    (or (member curpos (inherit-filler 'optionals ipname))
        (predict-next (1+ curpos) ipname length))
    (and *verbose*
      (st (format nil "~-s = ~s" (print-pattern (1+ curpos) length ipname)
        (inherit-filler 'target ipname)))))

; Function predict-next predicts the next element or elements; if there
; are optional elements, all of those up to the first non-optional element
; are predicted, as is that first non-optional one.
(defun predict-next (curpos ipname length)
  (let ((optionals (inherit-filler 'optionals ipname)))
    (do ((pos curpos (1+ pos)))
        ((or (> pos length)
             (not (member pos optionals)))
         (self curpos pos)
         (predict-next-element (inherit-filler curpos ipname) ipname)))
    (predict-next-element (inherit-filler options ipname) ipname))

; Function predict-next-element Predicts the next element of an IP. If that
; element is a word, the associated lexical node is predicted; otherwise it
; is the role filler for the given slot that is predicted. The functions
; predict-word and predict-role are in the file predictions.lsp.
(defun predict-next-element (elem ipname)
  (cond
    ((atom elem) (predict-word elem ipname))
    (t (predict-role (caddr elem) ipname)))

; Function fill-last-role checks to see if the last element of the phrase
; thus far recognized is a slot or a word. If it is a word, nothing needs
; to be done, so the pattern is returned as is. Otherwise, the
; pattern is returned as is.

```

```

filler for the given slot is recorded. The filler is the most recently
activated concept for the filler of the slot in the ip-target's ncp.
-- flag: the last element of a phrase does not have the same adjacency
  checks as intermediate elements; flag records this fact

(defun fill-last-role (ipname curpos ipmarker flag)
  (cond
    ; if this is the storage element, don't need to fill in prior slot
    ; (in fact, it may not exist, if the first elem is the storage elem
    (null curpos)
      (t (let* ((elem (inherit-filler curpos ipname))
                ; determine appropriate marker (such that adjacency
                ; constraints are maintained)
                (marker (cond
                  (flag
                    (scan-markers-for-fit elem ipmarker ipname)
                    (t (car (mop-ams (name->mop
                      (cond
                        ((atom elem) elem)
                        ((t (caddr elem)))))))
                    (replace-slot ipname 'current (1+ (am-finish marker)))
                    (cond
                      ; if it is a word, do nothing
                      ((atom elem))
                      ; otherwise, fill in the slot filler
                      ; (t
                        (add-role-filler curpos ipname
                          (list (car elem) (cadr elem) (am-parent marker))))))
                  (cond
                    ; This function returns the pattern with a * denoting the current position
                    (defun get-ip-slots (ipname)
                      (for (k :in (make-m-n 1 (inherit-filler 'length ipname)))
                        :when (let ((filler (role-filler k ipname)))
                                (and filler
                                      (listp filler)))
                        :save (cdr (role-filler k ipname)))
                    )
                    (defun print-pattern (curpos length ipname)
                      (append (for (k :in (make-m-n 1 length))
                        :when (< k curpos)
                        :save (let ((filler (inherit-filler k ipname)))
                                (cond
                                  ((atom filler) filler)
                                  ((t (cdr filler))))))
                    '(*)
                    (for (k :in (make-m-n 1 length)
                      :when (>= k curpos)
                      :save (let ((filler (inherit-filler k ipname)))
                            (cond
                              ((atom filler) filler)
                              ((t (cdr filler)))))))
                    )
                    ; This scans the nams for a target node to determine if one of them fits
                    ; adjacency requirements for the phrase. The first such am that fits
                    ; returned. The adjacency requirements are:
                    ; 1. the start of the current marker must be 1+ the finish of the previous
                    ; slot filler
                    ; 2. the start of the prior slot filler must be the position after
                    ; last filled slot (denoted by current)
                    ; This should all be recorded much more elegantly later, since it is rather
                    ; ugly.
                  )
                )
              )
            )
          )
        )
      )
    )
  )
)
```

; confusing as it stands now.

```

.....(defun fill-pre-store-slots (ipname)
  (let ((pre-slots (make-m-n (l- (inherit-filler 'store ipname)) 1))
        (optionals (inherit-filler 'optionals ipname))
        (all-filled t))
    ; assume all pre-store slots can be filled; if we get to one that can't
    ; be filled, invalidate that assumption
    (for (slot :in pre-slots)
         :when (and (not (fill-slot ipname slot))
                     (not (member slot optionals)))
         :do (setf all-filled nil))
    all-filled)

  ; Function fill-slot tries to fill in the given slot. If successful, t is
  ; returned, otherwise nil.
  (defun fill-slot (ipname pos)
    (let* ((filler (inherit-filler pos ipname))
           ; elem is the target slot filler
           (elem (cond
                    (atom filler) filler)
                    (t (caddr filler)))) ; see if an activation marker exists for the target slot filler
      ; which fulfills the adjacency requirements of the phrase
      ; (marker (find-marker elem (inherit-filler 'start ipname)))
      (cond
        ; if no marker can be found, then the slot filler was not recognized
        ; from the phrase; return nil
        ((null marker) nil)
        ; if the marker was found, and this is a word, the just update the
        ; ip start slot to be the start pos of this word
        ; (atom filler)
        (replace-slot ipname 'start (am-start marker))
        ; otherwise, fill the slot, and update start as above
        (t
          (replace-slot ipname pos (list (car filler) (cadr filler))
                        (am-parent marker)))
          (replace-slot ipname 'start (am-start marker)))))

  ; Function find-marker scans the markers for the target slot and returns the
  ; first one which fulfills the phrase adjacency requirement. This requirement
  ; specifies that the slot has to end just prior to the beginning of the current
  ; start slot of the ip.
  (defun find-marker (elem start)
    (for (am :in (mop-ams (name->mop elem)))
         :first (equal (am-finish am) (l- start))
         :do am))

  ; Function try-to-advance-ip checks to see if the ip is advanceable (i.e.
  ; the current am for the ip matches the next element (if a word) or is an
  ; abstraction of the slot filler for the next element (if not a word)).
  (defun try-to-advance-ip (ipname mop)
    (let* ((ipmarker (car (mop-ams mop)))
           (curpos (find-curpos ipmarker ipname mop)))
      (cond (curpos
              (insist try-to-advance-ip curpos)
              ; if this ip just got activated (as determined by termflag), then
              ; activate its terminator node
              (cond
                (null (role-filler 'termflag ipname))
                (replace-slot ipname 'termflag t)
                (activate (inherit-filler 'terminator ipname) nil)
                (make-am :parent ipname :owners (list ipname) nil)))
            .....(cond
                  ((advanceablelp (inherit-filler curpos ipname) 'ipmarker ipname curpos)
                   (advance-ip ipname mop ipmarker curpos
                               (inherit-filler 'length ipname)))
                  ; if not advanceable, remove the index pattern (or at least do
                  ; *something* with the ip
                  (t
                    (and *debug*
                        (format t "~>Warning! null CURPOS in TRY-TO-ADVANCE"))))
                  (t (and *debug*
                          (format t "~>Note: advance-ip failed -- repredicting ~S." ipname))
                      ; may want to delete the ip here, instead of repredicting it,
                      ; once the kill order is better specified
                      (predict-next-element (inherit-filler curpos ipname)))
                    (remove-mop ipname)))
                  ; Since some elements of a phrase may be optional, find-curpos determines
                  ; which position in the phrasal pattern had its prediction fulfilled. It
                  ; simply steps through the slots from index on until it finds a match for
                  ; the activated node (as specified through ipmarker).
                  (defun find-curpos (ipmarker ipname mop)
                    (let ((curpos (inherit-filler 'index ipname)))
                      (for (l :in (make-m-n curpos (inherit-filler 'length ipname)))
                           :first (let ((filler (inherit-filler i ipname)))
                                    (cond
                                      ((atom filler)
                                       (and (abstp (name->mop filler)
                                                   (name->mop (am-parent ipmarker)))
                                            1))
                                      (t
                                        (and (abstp (name->mop (caddr filler)
                                                    (name->mop (am-parent ipmarker)))
                                                    1)))))))
                        ; Advanceablelp checks to see if the next element matches the last activated
                        ; node or is an abstraction of the last activated node. This determines
                        ; whether the pattern can be advanced.
                        (defun advanceablelp (curelem ipname curpos)
                          (cond
                            ; if this is not the storage element in the pattern, make sure adjacency
                            ; requirements of the ip are met
                            ; (and (not (equal curpos (inherit-filler 'store ipname)))
                            ;       (let* ((nevp (inherit-filler 'fill-next ipname))
                            ;             (newelem (inherit-filler newpos ipname)))
                            ;               (not (scan-markers-for-fit newelem ipmarker ipname))))
                            ;       n1)
                            (atom curelem)
                            (equal (am-parent ipmarker) curelem)
                            (t
                              (abstp (name->mop (caddr curelem))
                                     (name->mop (am-parent ipmarker))))))

                          ; Function advance-ip advances the ip (amazing!). When a pattern is advanced,
                          ; the information (slot filler) for the last element is supplied by appending
                          ; it to that slot in the ip. So, an element of the ip is not truly recognized
                          ; until the next element is referenced. This prevents premature binding of a
                          ; slot, by allowing subsequent (higher-order) processing to occur.
                          ; If there is still more of the pattern left, the next element in the ip is
                          ; predicted. Otherwise, since the pattern is completed, the target node is
                          ; activated and the ip destroyed.
                          (defun advance-ip (ipname mop ipmarker curpos length)
                            ; record the slot filler for the previous element
.....
```

```

.....(defun fill-last-role ipname (inherit-filler 'fill-next ipname) ipmarker t)
  ; if this is the storage element, don't need to fill in prior slot
  ; (in fact, it may not exist, if the first elem is the storage elem)
  ; increment index to next element
  (cond
    (equal curpos length)
      ; the pattern is finished -- recognize it
      ; first, record the slot filler for the last element
      ; (null curpos)
      ; (let* ((elem (inherit-filler curpos ipname))
      ;       (t (marker (cond
      ;           ; determine appropriate marker (such that adjacency
      ;           ; constraints are maintained)
      ;           (marker (cond
      ;               (flag
      ;                   (scan-markers-for-fit elem ipmarker ipname)
      ;                   (t
      ;                       (car (mop-ams (name->mop
      ;                           (atom elem) elem)
      ;                               (t (caddr elem)))))))
      ;               (replace-slot ipname 'current (1+
      ;                   (mop-ams (name->mop
      ;                       (atom elem) elem)
      ;                           (t (am-finish marker))))))
      ;               (replaced ipname 'current (1+
      ;                   (mop-ams (name->mop
      ;                       (atom elem) elem)
      ;                           (t (caddr elem)))))))
      ;               (cond
      ;                   ; if it is a word, do nothing
      ;                   (atom elem)
      ;                   ; otherwise, fill in the slot filler
      ;                   (t
      ;                       (add-role-filler curpos ipname
      ;                           (list (car elem) (cadr elem) (am-parent marker)))))))
      ;           )))))
      ;       )));
      ;       ; Function get-ip-slots returns a list of all the slots found in the ip.
      ;       ; (defun get-ip-slots (ipname)
      ;       ;   (for (k :in (make-m-n 1 (inherit-filler 'length ipname)))
      ;       ;     (when (let ((filler (role-filler k ipname)))
      ;       ;           (and filler
      ;       ;               (listp filler)))
      ;       ;             :save (cdr (role-filler k ipname)))))

      ;       ; This function returns the pattern with a * denoting the current position.
      ;       ; (defun print-pattern (curpos length ipname)
      ;       ;   (append (for (k :in (make-m-n 1 length))
      ;       ;             :when (< k curpos)
      ;       ;               (let ((filler (inherit-filler k ipname)))
      ;       ;                 :save (cdr (role-filler k ipname)))))

      ;       ; This function predicts the next element or elements; if there
      ;       ; are optional elements, all of those up to the first non-optional element
      ;       ; are predicted, as is that first non-optional one.
      ;       ; (defun predict-next (curpos ipname length)
      ;       ;   (let ((optionals (inherit-filler 'optionals ipname)))
      ;       ;     (do ((pos curpos (1+ pos)))
      ;       ;         (not (member pos optionals)))
      ;       ;         (setf curpos pos)
      ;       ;         (predict-next-element (inherit-filler pos ipname) ipname)
      ;       ;         (predict-next-element (inherit-filler curpos ipname) ipname)
      ;       ;         (predict-next-element (inherit-filler pos ipname) ipname)
      ;       ;         (predict-next-element (inherit-filler curpos ipname) ipname)
      ;       ;         (predict-word ipname)
      ;       ;         (predict-role (caddr elem) ipname)))
      ;       ;     (predict-next-element (elem ipname)
      ;       ;       (cond
      ;       ;           ((atom elem) (predict-word elem ipname))
      ;       ;           (t (predict-role (caddr elem) ipname)))))

      ;       ; Function predict-next-element Predicts the next element of an IP. If that
      ;       ; element is a word, the associated lexical node is predicted; otherwise it
      ;       ; is the role filler for the given slot is predicted. The functions
      ;       ; predict-word and predict-role are in the file predictions.lsp.

      ;       ; Function fill-last-role checks to see if the last element of the phrase
      ;       ; thus far recognized is a slot or a word. If it is a word, nothing needs
      ;       ; to be done, so the pattern is returned as is. Otherwise, the
      ;       ; filler for the given slot is recorded. The filler is the most recently
      ;       ; activated concept for the filler of the slot in the ip-target's mop.
      ;       ; -- flag: the last element of a phrase does not have the same adjacency
      ;       ; checks as intermediate elements; flag records this fact
      ;       ; (defun fill-last-role (ipname curpos ipmarker flag)

```

```

.....(cond
    ; if this is the storage element, don't need to fill in prior slot
    ; (in fact, it may not exist, if the first elem is the storage elem)
    ; (null curpos)
    ; (let* ((elem (inherit-filler curpos ipname)))
    ;   (marker (cond
    ;       ; determine appropriate marker (such that adjacency
    ;       ; constraints are maintained)
    ;       (marker (cond
    ;           (flag
    ;               (scan-markers-for-fit elem ipmarker ipname)
    ;               (t
    ;                   (car (mop-ams (name->mop
    ;                       (atom elem) elem)
    ;                           (t (caddr elem)))))))
    ;               (replace-slot ipname 'current (1+
    ;                   (mop-ams (name->mop
    ;                       (atom elem) elem)
    ;                           (t (am-finish marker))))))
    ;               (replaced ipname 'current (1+
    ;                   (mop-ams (name->mop
    ;                       (atom elem) elem)
    ;                           (t (caddr elem)))))))
    ;               (cond
    ;                   ; if it is a word, do nothing
    ;                   (atom elem)
    ;                   ; otherwise, fill in the slot filler
    ;                   (t
    ;                       (add-role-filler curpos ipname
    ;                           (list (car elem) (cadr elem) (am-parent marker)))))))
    ;           )))))
    ;       ; Function get-ip-slots returns a list of all the slots found in the ip.
    ;       ; (defun get-ip-slots (ipname)
    ;       ;   (for (k :in (make-m-n 1 (inherit-filler 'length ipname)))
    ;       ;     (when (let ((filler (role-filler k ipname)))
    ;       ;           (and filler
    ;       ;               (listp filler)))
    ;       ;             :save (cdr (role-filler k ipname)))))

    ;       ; This function returns the pattern with a * denoting the current position.
    ;       ; (defun print-pattern (curpos length ipname)
    ;       ;   (append (for (k :in (make-m-n 1 length))
    ;       ;             :when (< k curpos)
    ;       ;               (let ((filler (inherit-filler k ipname)))
    ;       ;                 :save (cdr (role-filler k ipname)))))

    ;       ; This scans the ams for a target node to determine if one of them fits the
    ;       ; adjacency requirements for the phrase. The first such am that fits is
    ;       ; returned. The adjacency requirements are:
    ;       ; 1. the start of the current marker must be 1+ the finish of the prior
    ;       ; slot filler
    ;       ; 2. the start of the prior slot filler must be the position after the
    ;       ; last filled slot (denoted by current)
    ;       ; This should all be recorded much more elegantly later, since it is rather
    ;       ; confusing as it stands now.
    ;       ; (defun scan-markers-for-fit (elem ipmarker ipname)
    ;       ;   (for (marker :in (mop-ams (name->mop
    ;       ;       (atom elem) elem)
    ;       ;           (t (caddr elem))))))


```

```
(t (caddr elem)))))  
:first (and (equal (am-start ipmarker)  
 (l+ (am-finish marker)))  
 (equal (am-start marker)  
 (role-filler 'current ipname)  
 marker)))
```

```

***** FANSYS: A Computer Model of Text Comprehension and Question
***** Answering for Failure Analysis
*****
** File: hips.lsp
**
** Developed by: Sergio J. Alvarado
**                 Ronald K. Braun
**                 Kenrick J. Mock
**
** Artificial Intelligence Laboratory
** Computer Science Department
** University of California
** Davis, CA 95616
**
** Funds for the support of this study have been allocated by the
** NASA-Ames Research Center, Moffett Field, California, under
** Interchange No. NCA2-721.
****

; This file implements higher-level index patterns (hips). Hips are
; designed to bypass the adjacency requirements of normal index
; patterns, allowing the parser to look for related concepts spread
; throughout a text being parsed. As such, elements of a hip should
; be concepts only, not words. The concept of an hip might be
; unnecessary, since index patterns should really just be
; generalized to allow for relaxed adjacency constraints.

; The macro defpat defines a hip. The format of a hip is:
; (defpat target elem1 [elem2 ... elemn] :activator actmop
;       [::context context] [::mode mode])
; The target is the mop that should be activated if at least one of
; the elements has been activated, and the activator node has been
; activated.
; Elements are concepts that activate this hip. The hip itself is
; stored with all of the concepts in the elem-list, so activating any
; of them causes this pattern to be activated. Note that individual
; elements are optional in the sense that all of them need not appear
; for the pattern to be activated as whole. That is, the hip
; collects together any of the concepts that are activated; not all
; such concepts need to have occurred in the text.
; The context is the mop that wraps up processing of the hip. When
; the activator is activated during the parse, the hip is closed (so
; to speak) and the target mop activated with the slots given by the
; collected concepts. The activator is typically an end of text
; marker of some sort.
; The context is a contextual constraint on the hip that requires
; that the context be defined and active in order for this hip to be
; activated.
; The mode controls whether a hip is activate during a given
; mode; i.e. during question mode or parse mode.

(defmacro defpat (mopname &rest pattern-list)
  '(let* ((pattern ',pattern-list)
         (contextno nil)
         (modeno nil)
         (activatorno nil)
         (activator nil)
         (decompose out all of the hip specifiers in the hip definition
         (setf activatorno (position ',activator pattern)))

```

```

(activatorno
  (setf activator (nth (1+ activatorno) pattern))
  (setf (elt pattern activatorno) nil)
  (setf (elt pattern (1+ activatorno)) nil))
  (setf contextno (position ',context pattern))
  (cond
    (contextno
      (setf (elt pattern contextno) nil)))
    (setf modenr (position ',mode pattern))
    (cond
      (modeno
        (setf mode (nth (1+ modenr) pattern))
        (setf (elt pattern modenr) nil)
        (setf (elt pattern (1+ modenr)) nil))
        (setf pattern (remove-if #'(lambda (x) (equal nil x)) pattern))
        (cond
          (activator
            (setf (define-pattern ',mopname pattern activator mode contextno)
                  (t
                    (and *verbose*
                        (st (format nil "~>ignoring pattern ~s~s -- no activator!"'
                                      ',mopname ',pattern-list)))))))
          (Function define-pattern does the actual work of storing the pattern,
           ; creating the hip mop, and so forth.

          (defun define-pattern (mopname pattern activator mode contextno &aux newmop)
            (let ((pat (gentemp (format nil "~s." 'm-hl-pattern)))
                  (pat-slots nil))
                ; format the slots for a phrase in an atypical, but useful, manner
                ; for (k :in (make-n-1 (length pattern)))
                ;   do (let* ((elем (nth (1- k) pattern))
                ;             (fill) (role-filler elem mopname)))
                ;     (fill)
                ;   (and elem fill)
                ;   (setf pat-slots (cons (list k (list fill elem))
                ;                         pat-slots)))
                ; put the slots in a nice order
                (setf pat-slots (reverse pat-slots))
                (and *debug* (format t "&-s" pat-slots = ~s" pat-slots)))
                ; create the hip itself
                (setf newmop
                  (new-mop pat 'm-hl-pattern) 'mop (append pat-slots
                    `((target ,mopname) (activator ,activator) (mode ,mode)
                      (length ,(length pattern)) (termflag nil))))
                  ; create a context slot for this mop
                  (and contextno
                    (setf (mop-slots newmop) (cons ',(context m-root)
                      (mop-slots newmop)))
                    (cons pat (mop-slots newmop)))
                    (cons pat (mop-slots newmop)))))

          (deassoc fn for an hip. It
          ; behaves differently depending on which mop caused the activation of
          ; the hip. There are several such activating mops, including the
          ; activator, each of the elem concepts, and so on. There is also
          ; additional processing required the first time the hip is activated,
          ; as opposed to subsequent times.

```

```

(splice (for (equiv :in (mop-equivalent (name->mop)))
  (caadr (assoc elem (mop-slots mop)))))
  :save (list (caaddr (assoc elem (mop-slots mop))) equiv)))

(cond
  ; first make sure this hlip can be activated in this mode; if
  ; not, kill the hlip
  ((and (equal (mop-type mop) 'instance)
    (let ((filler (inherit-filler 'mode hname)))
      (and filler (not (equal filler *mode*)))))
   (remove-mop hname))
  (remove-mop hname)
  ; also make sure that if a contextual constraint has been
  ; specified for the hlip, we are in the right context;
  ; otherwise, kill the hlip
  ((equal hname 'm-hl-pattern)
   (and (equal (mop-type mop) 'instance)
     (member 'm-hl-pattern (mop-all-abssts mop))
     (inherit-filler 'context hname)
     (not (contextp (am-parent (car (mop-ams mop)))))))
  (remove-mop hname)
  ; don't do any processing for instance mops; only process the
  ; abstract ones, which organize each instance (which
  ; in turn correspond to each activation of an elem in the text)
  ((equal (mop-type mop) 'instance)
   (let ((target (target-filler 'target hname)))
     ; if the activator was responsible for activating this hlip,
     ; then close up the pattern and activate the target
     (abstp (name->mop (role-filler 'activator hname))
     (name->mop (am-parent (car (mop-ams mop))))))
     ; note that the pattern is no longer active
     (replace-slot hname 'termflag nil)
     (let ((slots nil)
       (setf slots (collect-all-slots mop hname)))
       (for (spec :in (mop-specs mop))
         :do (remove-mop spec)
         (let ((target (inherit-filler 'target hname)))
           (activate target slots nil t)))
         (activate target slots nil t)))
     ; force repeated activation
     (and (null (inherit-filler 'termflag hname))
       (mop-ams (name->mop 'm-context))
       (equal (get-counter)
         (ram-start (car (mop-ams (name->mop 'm-context))))))
     (replace-slot hname 'termflag nil)
     (let ((slots nil)
       (setf slots (collect-all-slots mop hname)))
       (for (spec :in (mop-specs mop))
         :do (remove-mop spec)
         (let ((target (inherit-filler 'target hname)))
           (activate target slots nil t)))
         (activate target slots nil t)))
     ; this is the first time this hlip has been activated by an
     ; elem in the pattern -- inform the activator
     ; termflag and predict the activator
     (null (inherit-filler 'termflag hname))
     (predict (role-filler 'activator hname) hname)
     (replace-slot hname 'termflag 't)))
   (Function collect-all-slots collects together all of the activated
   ; instances of a given hlip, pulls out the relevant activating
   ; concepts, and creates the slot list of activated concepts. Some
   ; handwaving is done to allow equivalent mops to be handled.

  (defun m-hl-pattern-in (hname)
    (let ((mop (name->mop hname)))
      (cond
        ; first make sure this hlip can be activated in this mode; if
        ; not, kill the hlip
        ((and (equal (mop-type mop) 'instance)
          (let ((filler (inherit-filler 'mode hname)))
            (and filler (not (equal filler *mode*)))))
         (remove-mop hname))
        (remove-mop hname)
        ; also make sure that if a contextual constraint has been
        ; specified for the hlip, we are in the right context;
        ; otherwise, kill the hlip
        ((equal hname 'm-hl-pattern)
         (and (equal (mop-type mop) 'instance)
           (member 'm-hl-pattern (mop-all-abssts mop))
           (inherit-filler 'context hname)
           (not (contextp (am-parent (car (mop-ams mop)))))))
        (remove-mop hname)
        ; don't do any processing for instance mops; only process the
        ; abstract ones, which organize each instance (which
        ; in turn correspond to each activation of an elem in the text)
        ((equal (mop-type mop) 'instance)
         (let ((target (target-filler 'target hname)))
           ; if the activator was responsible for activating this hlip,
           ; then close up the pattern and activate the target
           (abstp (name->mop (role-filler 'activator hname))
           (name->mop (am-parent (car (mop-ams mop))))))
           ; note that the pattern is no longer active
           (replace-slot hname 'termflag nil)
           (let ((slots nil)
             (setf slots (collect-all-slots mop hname)))
             (for (spec :in (mop-specs mop))
               :do (remove-mop spec)
               (let ((target (inherit-filler 'target hname)))
                 (activate target slots nil t)))
               (activate target slots nil t)))
             ; force repeated activation
             (and (null (inherit-filler 'termflag hname))
               (mop-ams (name->mop 'm-context))
               (equal (get-counter)
                 (ram-start (car (mop-ams (name->mop 'm-context))))))
             (replace-slot hname 'termflag nil)
             (let ((slots nil)
               (setf slots (collect-all-slots mop hname)))
               (for (spec :in (mop-specs mop))
                 :do (remove-mop spec)
                 (let ((target (inherit-filler 'target hname)))
                   (activate target slots nil t)))
                 (activate target slots nil t)))
               ; this is the first time this hlip has been activated by an
               ; elem in the pattern -- inform the activator
               ; termflag and predict the activator
               (null (inherit-filler 'termflag hname))
               (predict (role-filler 'activator hname) hname)
               (replace-slot hname 'termflag 't)))
             (Function collect-all-slots collects together all of the activated
             ; instances of a given hlip, pulls out the relevant activating
             ; concepts, and creates the slot list of activated concepts. Some
             ; handwaving is done to allow equivalent mops to be handled.

            (defun collect-all-slots (mop mophname)
              (let ((slots nil)
                (slot-list (for (elem :in (make-m-n 1 (role-filler 'length
                  mophname)))) :first (and (abstp (name->mop (name->mop
                    (assoc elem (mop-slots mop)))) equiv)))) :rest
                (list (car elem) specmop)))))))
```

```
*****  
/*  
 * FANSYS: A Computer Model of Text Comprehension and Question  
 * Answering for Failure Analysis  
 *  
 */  
  
/* File: hips.mops  
 */  
  
/* Developed by: Sergio J. Alvarado  
 * Ronald K. Braun  
 * Kenrick J. Mock  
 */  
  
/* Artificial Intelligence Laboratory  
 * Computer Science Department  
 * University of California  
 * Davis, CA 95616  
 */  
  
/* Funds for the support of this study have been allocated by the  
 * NASA-Ames Research Center, Moffett Field, California, under  
 * Interchange No. NCA2-721.  
 */  
  
*****  
*****  
/* This file organizes the mops necessary to implement higher-level index  
 * patterns (hips).  
 */  
  
(defmop m-hl-pattern (m-processing-mop))  
(add-assoc-fn m-hl-pattern m-hl-pattern-fn)  
;  
; see hips.lsp for the definition of m-hl-pattern-fn.
```

```

*****
;*
;* FANSYS: A Computer Model of Text Comprehension and Question
;* Answering for Failure Analysis
;*
;* File: failure.lsp
;* Developed by: Sergio J. Alvarado
;*                 Ronald K. Braun
;*                 Kendrick J. Mock
;*
;* Artificial Intelligence Laboratory
;* Computer Science Department
;* University of California
;* Davis, CA 95616
;*
;* Funds for the support of this study have been allocated by the
;* NASA-Ames Research Center, Moffett Field, California, under
;* Interchange No. NCA2-721.
;*
;* This file organizes repair strategies for failures. The idea of applying
;* failure repair strategies for parsing failures has not been at all
;* explored. Currently, only new words (or misspellings!) that cause a
;* parsing failure are repaired by adding the word to the lexicon.
;*
;* r-define-unknown-word is the strategy associated with f-unknown-word.
;* The strategy simply defines a new lexical node for the unknown word,
;* indexing it under m-unknown-word.
;*
(defun r-define-unknown-word (mopname)
  (let ((mop (name->mop mopname)))
    (word (role-filler 'word mopname)))
  (cond
    ((listp-mopp mop)
     (and *verbose*
          (st (format nil "~& Failure: ~s is an unknown-word." word)))
      (and *verbose*
           (st (format nil "~& Repair: define-unknown-word on ~s." word)))
      (new-mop word '(m-unknown-word) 'instance nil)
      (remove-mop mopname)))
    
```

```
*****
;* FANSYS: A Computer Model of Text Comprehension and Question
;* Answering for Failure Analysis
;*
;* File: failure.mops
;*
;* Developed by: Sergio J. Alvarado
;*                 Ronald K. Braun
;*                 Kenrick J. Mock
;*
;* Artificial Intelligence Laboratory
;* Computer Science Department
;* University of California
;* Davis, CA 95616
;*
;* Funds for the support of this study have been allocated by the
;* NASA-Ames Research Center, Moffett Field, California, under
;* Interchange No. NCA2-721.
;*
;* This file organizes mops which represent failures during the parsing
;* and comprehension process.
;*
(defmop m-failure (mr-processing-mop))
; Failure f-unknown-word is instantiated when an unknown word is encountered
; during parsing.
(defmop f-unknown-word (m-failure) (word nil))
(add-assoc-fn f-unknown-word r-detect-unknown-word)
```

```

(AND WHEN-PART (CAR (CDR WHEN-PART))
  (FOR-BODY FOR-CLUSES)))
  (DEFUN FOR-VAR-FORMS (L)
    (AND L (LISTP (CAR L))
      (CONS (CAR L) (FOR-VAR-FORMS (CDR L)))))

  (DEFUN FOR-BODY (L)
    (AND L (OR (AND (FOR-KEY (CAR L)) L)
      (FOR-BODY (CDR L)))))

  (DEFUN FOR-EXPANDER (VAR-FORMS WHEN-FORM BODY-FORMS)
    (INSIST FOR
      (NOT (NULL VAR-FORMS))
      (NOT (NULL BODY-FORMS)))
    (LET ((VARS (MAPCAR #'(LAMBDA (VAR-FORM)
      (CAR (CDR (CDR VAR-FORM)))) (CAR (CDR (CDR (CDR VAR-FORM)))))))
      (MAPCN-BODY (FUNCALL (FOR-KEY (CAR BODY-FORMS)
        WHEN-FORM
        `(PROGN ,@ (CDR BODY-FORMS))))))

      `,(CAR MAPCN-BODY)
      #'(LAMBDA ,VARS ,(CAR (CDR MAPCN-BODY)))
      ,@LISTS)))
    (DEFMACRO DEFINE-FOR-KEY (KEY VARS MAPFN BODY)
      `(PROGN (SETF (FOR-KEY ',KEY)
      #'(LAMBDA ,VARS (LIST ,MAPFN ,BODY))
      ,KEY))

      (DEFINE-FOR-KEY :ALWAYS (TEST BODY)
        'EVERY
        (COND (TEST `(OR (NOT ,TEST) ,BODY)) (T BODY)))
      (DEFINE-FOR-KEY :DO (TEST BODY)
        'MAPC (COND (TEST `(AND ,TEST ,BODY)) (T BODY)))
      (DEFINE-FOR-KEY :FILTER (TEST BODY)
        'MAPCAN
        (LET ((FBODY `(LET ((X ,BODY)) (AND X (LIST X)))))
          (COND (TEST `(AND ,TEST ,FBODY)) (T FBODY))))
      (DEFINE-FOR-KEY :FIRST (TEST BODY)
        'SOME (COND (TEST `(AND ,TEST ,BODY)) (T BODY)))
      (DEFINE-FOR-KEY :SAVE (TEST BODY)
        (COND (TEST 'MAPCAN) (T 'MAPCAR)
        (COND (TEST `(AND ,TEST (LIST ,BODY)))
        (T BODY)))))

      (DEFUN FOR-KEY :SPLICED (TEST BODY)
        'MAPCAN
        '(COPY-LIST
        , (COND (TEST `(AND ,TEST ,BODY)) (T BODY)))))
  
```

```

* * * * *
* File: lisp_utils.lsp
* * * * *
* Developed by: Sergio J. Alvarado
* Ronald K. Braun
* Kenrick J. Mock
* * * * *
* Artificial Intelligence Laboratory
* Computer Science Department
* University of California
* Davis, CA 95616
* * * * *
* Funds for the support of this study have been allocated by the
* NASA-Ames Research Center, Moffett Field, California, under
* Interchange No. NCA-2-721.
* * * * *
* This file contains general lisp utilities and extensions.
* * * * *
; The following functions are unmodified from Inside Case-Based
; Reasoning (ICBR) by Riesbeck and Schank. See ICBR for documentation

(DEFMACRO INSIST (FNAME &REST EXPS)
  '(AND ,@ (MAKE-INSIST-FORMS FNAME EXPS)))
(DEFUN MAKE-INSIST-FORMS (FNAME EXPS)
  (AND (NOT (NULL EXPS))
       (CONS 'OR ' (CAR EXPS))
       (PROGN (DEFUN FN (&REST EXPS)
                     (ERROR "<S> failed in ~S"
                           ,(CAR EXPS) ',FNNAME))
              (MAKE-INSIST-FORMS FNAME (CDR EXPS)))))

(DEFMACRO DEFINE-TABLE (FN VARS PLACE)
  (LET ((KEY (CAR VARS))
        (SET-FN (GENTEMP "SET-FN."))
        (VAL (GENTEMP "VAL.")))
    `(PROGN (DEFUN ,FN (&KEY) (GETF ,PLACE ,KEY))
           (DEFUN ,SET-FN (&KEY ,VAL)
                 (SETF (GETF ,PLACE ,KEY) ,VAL)
                 (DEFSETF ,FN ,SET-FN)
                 ,FN)))
(DEFUN DELETE-KEY (TABLE KEY)
  (REMFF TABLE KEY) TABLE)
(DEFUN TABLE-KEYS (TABLE)
  (AND TABLE
       (CONS (CAR TABLE)
             (TABLE-KEYS (CDR TABLE)))))

(SETF *FOR-KEYS* NIL)
(DEFINE-TABLE FOR-KEY (KEY *FOR-KEYS*)
(DEFMACRO FOR (&REST FOR-CLAUSES)
  (LET ((WHEN-PART (MEMBER '&WHEN FOR-CLAUSES))
        (WHEN-EXPANDER (FOR-VAR-FORMS FOR-CLAUSES)))
    `(PROGN ,WHEN-PART
           (WHEN ,WHEN-EXPANDER
             ,FOR-CLAUSES)))

```

```

(defmacro show (&rest namelist)
  `',(for (name :in ',namelist)
         :do (format t "&~<-~>~%" (name->mop name)))))

; This macro stores the given assoc-fn under the given mop.

(defmacro add-assoc-fn (mopname fname)
  `'(add-assoc-fn-fn ,mopname ,fname))

; Function add-assoc-fn-fn does the actual work of storing the function
; with the mop as an associated function.
; Use the no-inherit flag if you don't want the assoc fn propagated down
; to the specs of the mop mopname.

(defun add-assoc-fn-fn (mopname fname &optional no-inherit)
  (let ((mop (name->mop mopname)))
    (cond
      ((not (member fname (mop-assoc-fns mop)))
       (setf (mop-assoc-fns mop) (cons fname (mop-assoc-fns mop))))
      (if (not no-inherit)
          (for (spec :in (mop-specs mop))
            (do (add-assoc-fn-fn spec fname)))))))

; Determines the current counter in the text from m-done-with-word.

(defun get-counter ()
  (role-filler 'count 'm-done-with-word))

; Convenience macro to look at a function with maximal ease.

(defmacro sf (fn)
  `(symbol-function ',fn))

*****  

; These are miscellaneous utility functions. Except for convenience
; functions, these should probably be migrated into other files.

; This function replaces the contents of a slot in a given mop.
; This is crudely accomplished by deleting the old slot and
; adding a new one with the desired content. This crudeness is necessary
; in terms of creating copies of lists for replacement; replacing in existing
; lists tends to follow internal links back to unpredictable places.

(defun replace-slot (mopname role filler)
  (let ((mop (name->mop mopname)))
    (setf (mop-slots mop)
          (remove (role-slot role mopname) (mop-slots mop)))
    (add-role-filler role mopname filler)))

; This predicts the targetmop for mop mopname. (Or something like that...)

(defun predict (mopname targetmop)
  `(<< modified 11/29/92 by r ***equivalence**'
     (for (name :in (mop-equivalent (name->mop mopname)))
       (do (let ((mop (name->mop name))
                (pm (make-pm :parent targetmop))
                (setf (mop-pms mop) (cons pm (mop-pms mop)))
                pm)))
        ;>>>
        ; Function remove-am removes the given activation marker from the parent
        ; node and all abstractions.

        (defun remove-am (am mopname)
          (for (abst :in (mop-all-absts (name->mop mopname)))
            (do (let ((mop (name->mop abst)))
                  (and *debug* (format t "&~<-~>~%" abst))
                  (setf (lam-owners am)
                        (remove abst (lam-owners am)))
                  (remove *debug* (format t "&~<-~>~%" am))
                  (and (mop (setf (mop-ams mop) (remove am (mop-ams mop)))))))
            (and *debug* (format t "&~<-~>~%" mop = "s" mop)))
        ; This macro converts a list of mopnames into mops and prints them.

```

```

*****+
/* FANSYS: A Computer Model of Text Comprehension and Question
   Answering for Failure Analysis
   *
   * File: Predictions.lsp
   *
   * Developed by: Sergio J. Alvarado
   *                 Ronald K. Braun
   *                 Kenrick J. Mock
   *
   * Artificial Intelligence Laboratory
   * Computer Science Department
   * University of California
   * Davis, CA 95616
   *
   * Funds for the support of this study have been allocated by the
   * NASA-Ames Research Center, Moffett Field, California, under
   * Interchange No. NCA2-721.
   *
*****+
; This file organizes the functions necessary for the implementation
; of the various prediction strategies. These strategies have been
; dramatically simplified over the original (in fact, they've been
; stripped away entirely). Predictions are simply left sitting around
; until fulfilled, at which time they are removed if the target mops
; are no longer around.

; Function predict-word simply predicts the target word.

(defun predict-word (word ipname)
  (and *debug* (format t "~&Predicting ~s: target = ~s" word ipname))
  (predict word ipname))

; Function predict-role simply predicts the target role map.

(defun predict-role (role-mop ipname)
  (and *debug* (format t "~&Predicting ~s: target = ~s" role-mop ipname))
  (predict role-mop ipname))

```

```

(replace-slot mopname 'active nil)
; case 1
  (abstop (name->mop 'm-index-pattern) (name->mop parent))
    (cond
      ; if active, set flag and predict terminating mop
      ((null (role-filler 'active mopname))
       (replaced-slot mopname 'active t)
       (predict (role-filler 'terminator mopname) mopname))
      (predict (role-filler 'preds
        (replace-slot mopname 'preds
          (cons parent (role-filler 'preds mopname))))))
      ; case 2
      (t
        (predict 'm-done-with-word mopname))))))

*****  

/* FANSYS: A Computer Model of Text Comprehension and Question  

 * Answering for Failure Analysis  

 */
File: terminators.lisp
/* Developed by: Sergio J. Alvarado
 * Ronald K. Braun
 * Kenrick J. Mock
 */
/* Artificial Intelligence Laboratory
 * Computer Science Department
 * University of California
 * Davis, CA 95616
 */

/* Funds for the support of this study have been allocated by the
 * NASA-Ames Research Center, Moffett Field, California, under
 * Interchange No. NCA2-721.
 */

*****  

; This file organizes the mops and functions used for killing ips
; when their termination mop is activated.

; A separate mop is created for each possible terminator (as encountered
; in defphrase defns). These mops are added at defphrase time.
; -- terminator : the actual name of the terminator mop
; -- index: a look-up table of (mop, termnop) pairs
; -- active: flag, set to t on first occurrence of an ip with this
;           terminator

(defmop m-terminator (m-processing-mop) (terminator nil) (index nil)
  (add-assoc-in m-terminator terminator fn))

; Specs of m-terminator have additional slots:
; -- preds: a list of all active (or once active) ips to be killed
; -- pred: at the occurrence of the terminator mop, then predict
; -- active: flag, set to t on first occurrence of an ip with this
;           terminator

; The flow of control for termination is:
; 1. If the activating node is an index-pattern, then
;    a. Set the active flag and predict the terminating mop for the
;       first activating ip only (as determined by the active flag)
;    b. Add the ipname to the preds list
; 2. If the activating node is the terminator mop, then predict
;    m-done-with-word; the idea here is to delay the killing of ips
;    until all other processing for the word is finished.
; 3. If the activating mop is m-done-with-word, go ahead and delete
;    all of the ips in the pred-list, and turn off the active flag

(defun terminator-fn (mopname)
  (let* ((mop (name->mop mopname))
         (parent (am-parent (car (mop-ams mop)))))
    (cond
      ; work is done only for instances of m-terminator
      ((instance-mopp mop)
       (cond
         ; case 3
         (equal parent 'm-done-with-word)
         (for (pred :in (role-filler 'preds mopname))
              :do (and (mopp (name->mop pred))
                       (remove-mop pred)))
         (replace-slot mopname 'preds nil)

```

```

(* FANSYS: A Computer Model of Text Comprehension and Question
   Answering for Failure Analysis

File: generator.lsp

Developed by: Sergio J. Alvarado
               Ronald K. Braun
               Kenrick J. Mock
               Artificial Intelligence Laboratory
               Computer Science Department
               University of California
               Davis, CA 95616

Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NAG-2-721.

This file organizes the code necessary to generate from memory into
target natural language.

*generate* holds the result of the generation of the mop; this generation
is stored as a list of strings.

(let* ((mop (name->mop mopname))
      (ip (name->ip-ip-refs (name->mop abst)))
      (ip (for (abst :in (mop-all-absts mop))
              :first (for (ipname :in (mop-ip-refs (name->mop abst)))
                         :first (cond
                                  (t (and (abstP (name->mop constraint)
                                         (name->mop ipname))
                                         ipname)))))))
  (cond
    ; if there are no ips, just print the raw mopname, for lack of
    ; anything better to do
    (null ip)
    (setf *generate*
          (append *generate*
                  (list (format nil " ~s" mopname)))))

    (for (k :in (make-m-n 1 (inherit-filler 'length ip)))
         :do (let ((elem (inherit-filler k ip)))
               (cond
                 ; words get printed
                 ((atom elem)
                  ; print out punctuation nicely
                  (equal elem '*quote*')

```

```

***** *left-parent* B *right-parent* Mission Support *colon* None *period*
***** *left-parent* C *right-parent* System *colon* None *period*
***** *left-parent* D *right-parent* Interfaces *colon* None *period*
***** *eoc*)}

File: rc.1.txt
Developed by: Sergio J. Alvarado
Ronald K. Braun
Kenrick J. Mock
Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616
Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

This is an abstracted version of the first case description for the
ring concentrator. It is currently the only case that the parser can
handle. See the FEMA manuals for the full case description.

(seeif rc.1)

ITEM NAME *colon* Ring Concentrator

FAILURE MODE *colon* Loss of output - Failure to Start

FAILURE CAUSES *colon* Piece-part failures *comma* Contamination
*comma* Temperature *left-parent* High or Low *right-parent* *comma*
Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *slash* VERIFICATION *colon* Indication of a RC
failure is first detected by the *quote*
next *quote* active node on the network *period* System
Management
will reach a time-out limit for receipt of the network
token *period*.

CORRECTIVE ACTION *colon*

*left-parent* *right-parent* Short Term *colon* Network
reconfiguration is effected automatically *period* The DMS network
remains in operation in a reconfigured state with the failed RC
bypassed *period*

*left-parent* b *right-parent* Long Term *colon* the crewmen check
for *quote* applied
power *quote* and the crewmen check for *quote* connector tightness *quote*
*period* If the RC cannot then be placed in operation *comma* the rc is
removed and replaced with an ORU logistics spare *period*.

FAILURE EFFECT ON *colon*

*left-parent* A *right-parent* Crew *slash* SSE *colon* None *period*
The conditions associated with the replacement
of a RC within the network configuration are such that the network
is already in a reconfigured state supplying full services
*period*

```

```
(defphrase m-root DMS :gen)
```

```
*****  
/*  
 * FANSYS: A Computer Model of Text Comprehension and Question  
 * Answering for Failure Analysis  
 *  
 */  
  
File: entities.phrases  
/* Developed by: Sergio J. Alvarado */  
/* Developed by: Ronald K. Braun */  
/* Developed by: Kendrick J. Mock */  
/*  
 * Artificial Intelligence Laboratory  
 * Computer Science Department  
 * University of California  
 * Davis, CA 95616  
 *  
 * Funds for the support of this study have been allocated by the  
 * NASA-Ames Research Center, Moffett Field, California, under  
 * Interchange No. NCA2-721.  
 */  
*****  
;  
; Domain knowledge representation for the DMS of the Space Station Freedom:  
; Phrasal knowledge corresponding to the abstract DMS entities.  
;  
(defphrase m-heartbeat-message heartbeat message)  
(defphrase m-status-message status message)  
(defphrase m-erroneous-message erroneous message)  
(defphrase m-core-network-backup backup network)  
(defphrase m-generic-rc ring concentrator)  
(defphrase m-next-rc next active ring concentrator in the network)  
(defphrase m-backup-rc backup ring concentrator)  
(defphrase m-generic-gw gateway)  
(defphrase m-backup-gw backup gateway)  
(defphrase m-generic-sdp standard data processor)  
(defphrase m-backup-sdp backup standard data processor)  
(defphrase m-generic-tgu time generation unit)  
(defphrase m-backup-tgu backup time generation unit)  
(defphrase m-generic-msd mass storage device)  
(defphrase m-DMS data management system)  
(defphrase m-sm system management)  
(defphrase m-token network token :store 2)  
(defphrase m-crew Crew *slash* SSPE :store 3)  
(defphrase m-mission-support mission support)  
(defphrase m-systems system)  
(defphrase m-interfaces interfaces)  
(defphrase m-crew the crewmen :store 2)  
(defphrase m-power-system *quot* applied power *quot* :store 2)  
(defphrase m-connectors *quot* connector tightness *quot* :store 2)  
(defphrase m-core-network DMS network :opt (1) :store 2)  
(defphrase m-crew crew)
```

```

*****+
** FANSYS: A Computer Model of Text Comprehension and Question
** Answering for Failure Analysis
** File: events.phrases
** Developed by: Sergio J. Alvarado
**                 Ronald K. Brain
**                 Kenrick J. Mock
**                 Artificial Intelligence Laboratory
**                 Computer Science Department
**                 University of California
**                 Davis, CA 95616
** Funds for the support of this study have been allocated by the
** NASA-Ames Research Center, Moffett Field, California, under
** Interchange No. NCA-2721.
*****+
; Domain knowledge representation for the DMS of the Space Station Freedom:
; Phrasal knowledge corresponding to the abstract representation of events.

(defphrase m-null-event there is no effect *period*)

(defphrase m-send-message-event the (sender) sends a (message) to the
(recipient) *period* :store 3)

(defphrase m-receive-message-event the (recipient) receives a (message)
from the (sender) *period* :store 3)

(defphrase m-prepare-to-receive-event the (recipient) prepares to receive a
(message) from the (sender) *period* :store 3)

(defphrase m-wait-to-receive-event the (recipient) waits for a (message)
from the (sender) *period* :store 3)

(defphrase m-wait-to-transmit-event the (sender) waits to transmit a
(message) to the (recipient) *period* :store 3)

(defphrase m-time-out-event (recipient) will reach a time-out limit
for receipt of the (message) *period* :store 5)

(defphrase m-power-up-event the (actor) powers up the (object) *period*
:store 3)

(defphrase m-power-down-event the (actor) powers down the (object)
*period* :store 3)

(defphrase m-do-post-event the (actor) performs a power on self test
*period* :store 8)

(defphrase m-do-ecc-event the (actor) performs single bit error detection
and correction *comma* and two bit error detection on the (object)
*period* :store 6)

(defphrase m-logical-disconnect-event the (actor) logically disconnects
the (object) from the (object2) *period* :store 4)

(defphrase m-logical-connect-event the (actor) logically connects the
(object1) to the (object2) *period* :store 4)

```

```

(defphrase m-physical-disconnect-event the (actor) physically disconnects
the (object1) from the (object2) *period* :store 4)

(defphrase m-physical-connect-event the (actor) physically connects the
(object1) to the (object2) *period* :store 4)

(defphrase m-look-up-replacement-event the (actor) selects a (object)
to replace the (backed-up) from a look-up table maintained within
the DMS *period* :store 12)

(defphrase m-verify-event (actor) check for (object) :store 2 :nogen)

```

```
*****  
/* FANSYS: A Computer Model of Text Comprehension and Question  
/* Answering for Failure Analysis  
/*  
/* File: procedures.phrases  
/*  
/* Developed by: Sergio J. Alvarado  
/* Ronald K. Braun  
/* Kendrick J. Mock  
/*  
/* Artificial Intelligence Laboratory  
/* Computer Science Department  
/* University of California  
/* Davis, CA 95616  
/*  
/* Funds for the support of this study have been allocated by the  
/* NASA-Ames Research Center, Moffett Field, California, under  
/* Interchange No. NCA2-721.  
/*  
/* Domain knowledge representation for the DMS of the Space Station Freedom:  
/* Phrasal knowledge corresponding to sequences of events and procedures.  
/*  
/******  
(defphrase m-procedure (sequence-of-steps) :gen)
```

```
*****
;* FANSYS: A Computer Model of Text Comprehension and Question
;* Answering for Failure Analysis
;*
;* File: groups.phrases
;*
;* Developed by: Sergio J. Alvarado
;*                 Ronald K. Braun
;*                 Kenrick J. Mock
;*
;* Artificial Intelligence Laboratory
;* Computer Science Department
;* University of California
;* Davis, CA 95616
;*
;* Funds for the support of this study have been allocated by the
;* NASA-Ames Research Center, Moffett Field, California, under
;* Interchange No. NCA2-721.
;*
*****
```

```
; Domain knowledge representation for the DMS of the Space Station Freedom:
; Phrasal knowledge corresponding to the abstract representation of groups.
;*
(defphrase m-and-group (1) (2) (3) (4) (5) :opt (1 2 3 4 5) :gen)

(defphrase m-or-group options consist of *colon* *left-paren* A
*right-paren* (1) or *comma* *left-paren* B *right-paren* (2) :gen)
```

```
*****  
/*  
 * PANSYS: A Computer Model of Text Comprehension and Question  
 * Answering for Failure Analysis  
 */  
  
/* File: causes.phrases  
 */  
  
/* Developed by: Sergio J. Alvarado  
 * Ronald K. Braun  
 * Kenrick J. Mock  
 */  
  
/* Artificial Intelligence Laboratory  
 * Computer Science Department  
 * University of California  
 * Davis, CA 95616  
 */  
  
/* Funds for the support of this study have been allocated by the  
 * NASA-Ames Research Center, Moffett Field, California, under  
 * Interchange No. NCA2-721.  
 */  
  
/* Domain knowledge representation for the DMS of the Space Station Freedom:  
 * Phrasal knowledge corresponding to failure modes.  
 */  
  
(defphrase m-faulty-component-cause contamination)  
  
(defphrase m-contamination-cause temperature *left-paren*  
  (defphrase m-temperature-out-of-range-cause temperature *left-paren*  
    high or low *right-paren*)  
  (defphrase m-mechanical-shock-cause mechanical shock)  
  (defphrase m-thermal-shock-cause thermal shock)  
  (defphrase m-erroneous-input erroneous input)
```

```
*****  
/*  
 * FANSYS: A Computer Model of Text Comprehension and Question  
 * Answering for Failure Analysis  
 *  
 */  
/* File: modes.phrases  
 */  
/* Developed by: Sergio J. Alvarado  
 * Ronald K. Braun  
 * Kendrick J. Mock  
 */  
/* Artificial Intelligence Laboratory  
 * Computer Science Department  
 * University of California  
 * Davis, CA 95616  
 */  
/* Funds for the support of this study have been allocated by the  
 * NASA-Ames Research Center, Moffett Field, California, under  
 * Interchange No. NCA2-721.  
 */  
/* Domain knowledge representation for the DMS of the Space Station Freedom:  
 * Phrasal knowledge corresponding to the abstract DMS modes.  
 */  
(defphrase m-startup-no-output loss-of-output - failure-to-start)  
(defphrase m-operational-no-output loss-of-output - failure-during  
operation)  
(defphrase m-operational-erroneous-output erroneous-output - failure  
during operation)
```

```
*****
;* FANSYS: A Computer Model of Text Comprehension and Question
;* Answering for Failure Analysis
;*
;* File: detections.phrases
;* Developed by: Sergio J. Alvarado
;*                 Ronald K. Braun
;*                 Kendrick J. Mock
;*
;* Artificial Intelligence Laboratory
;* Computer Science Department
;* University of California
;* Davis, CA 95616
;*
;* Funds for the support of this study have been allocated by the
;* NASA-Ames Research Center, Moffett Field, California, under
;* Inter-change No. NCA2-721.
;*
;* ****
;* Domain knowledge representation for the DMS of the Space Station Freedom:
;* Phrasal knowledge corresponding to the abstract representation of failure
;* detection procedures.
;*
(defphrase m-detection-and-group (1) (2) (3) (4) (5) :opt (1 2 3 4 5) :gen)
(defphrase m-failure-detection detection procedure *colon*
(sequence-of-steps))
```

```
*****  
/* FANSYS: A Computer Model of Text Comprehension and Question  
/* Answering for Failure Analysis  
/*  
/* File: corrections.phrases  
/*  
/* Developed by: Sergio J. Alvarado *  
/* Ronald K. Braun *  
/* Kenrick J. Mock *  
/*  
/* Artificial Intelligence Laboratory *  
/* Computer Science Department *  
/* University of California *  
/* Davis, CA 95616 *  
/*  
/* Funds for the support of this study have been allocated by the *  
/* NASA-Ames Research Center, Moffett Field, California, under *  
/* Interchange No. NCA2-721. *  
/*  
*****  
;  
; Domain knowledge representation for the DMS of the Space Station Freedom:  
; Phrasal knowledge corresponding to the failure correction procedures.  
(defphrase m-correction-and-group (1) (2) (3) (4) (5) :opt (1 2 3 4 5) :gen)  
(defphrase m-short-term short term)  
(defphrase m-long-term long term)  
(defphrase m-failure-correction (framework) correction procedure *colon*  
(sequence-of-steps) :store 3 :opt (1))  
(defphrase m-undefined-and-group this procedure is not currently defined  
* period* :gen)
```

```
*****
;* FANSYS: A Computer Model of Text Comprehension and Question
;* Answering for Failure Analysis
;*
;* File: effects.phrases
;* Developed by: Sergio J. Alvarado
;*                 Ronald K. Braun
;*                 Kenrick J. Mock
;*
;* Artificial Intelligence Laboratory
;* Computer Science Department
;* University of California
;* Davis, CA 95616
;*
;* Funds for the support of this study have been allocated by the
;* NASA-Ames Research Center, Moffett Field, California, under
;* Interchange No. NCA2-721.
;*
;* Domain knowledge representation for the DMS of the Space Station Freedom:
;* Phrasal knowledge corresponding to failure effects.
;*
;* (defphrase m-failure-effect failure effect on the (effected-component)
;*      *colon* (sequence-of-steps) :score 2)
```

; the different contexts are enumerated here

```
*****  
;  
; FANSYS: A Computer Model of Text Comprehension and Question  
; Answering for Failure Analysis  
;  
; File: vocab.phrases  
;  
; Developed by: Sergio J. Alvarado  
;                 Ronald K. Brun  
;                 Kentick J. Mock  
;  
;                 Artificial Intelligence Laboratory  
;                 Computer Science Department  
;                 University of California  
;                 Davis, CA 95616  
;  
;  
; Funds for the support of this study have been allocated by the  
; NASA-Ames Research Center, Moffett Field, California, under  
; Interchange No. NCA2-721.  
*****  
  
;  
; This file organizes the mops, phrases, and functional knowledge  
; necessary to parse the first case for the ring concentrator.  
;  
; Much of the code here is first-pass material and is not documented  
; in-line. Further iterations and refinements should allow  
; generalizations to be made over this code, such that much of it  
; can be reimplemented in a more rational and systematic manner.  
;  
(defpat m-case failed-component mode cause detection correction effect  
  :activation *eoc* :mode case :context)  
  
(add-assoc-fn m-context m-context-fn)  
  
;  
; Function m-context-fn stores the specific context specialization  
; that activated m-context. The prior context is removed. The  
; m-context mop is consulted when determining what the current context  
; is for hilips and such.  
;  
(defun m-context-fn (mopname)  
  (let ((pm (name->mop mopname)))  
    (cond  
      ((equal 'instance (mop-type mop))  
       (let ((pms (mop-pms (name->mop 'm-context)))  
             (hold nil)  
             (temp nil))  
         (setf temp (cons pm temp))  
         (setf hold (cons pm hold))))  
      (t  
        (setf hold (cons pm hold))))  
    ))  
;  
; store the new context  
; (setf (mop-pms (name->mop 'm-context)) (append hold temp))  
;  
; remove prior context  
;(for (elem :in (mop-specs (name->mop 'm-context)))  
;     :first (or (null (role-filler 'context elem))  
;                 (abtp (name->mop (role-filler 'context elem))  
;                       (name->mop mopname))))  
;  
;;: ACTIONS  
;  
;;: CASES  
;  
;(do (remove-mop elem))
```

;; CAUSES

;; CORRECTIONS
(defphrase m-failure-correction (framework * colon* :nogen)

(defmop m-enumeration (m-processing-mop))

(defphrase m-enumeration *left-parent* a *right-parent* :store 2)

(defmop m-enumeration *left-parent* b *right-parent* :store 2)

(defmop m-enumeration *left-parent* c *right-parent* :store 2)

(defmop m-enumeration *left-parent* d *right-parent* :store 2)

(add-assoc-fn m-enumeration m-enumeration-fn)

; M-enumeration-fn rebuilds the current context for each of the
; enumerated items.

(defun m-enumeration-fn (mopname)

(let* ((mop (name->mop mopname)))

(cond

((instance-mopp mop)

(activate 'm-build-context (list (list 'context

(inherit-filler 'context (car (mop-specs

(name->mop m-context)))) nil t)))

(defphrase m-rc rc :nogen)

(defphrase m-replace-with-spare the (failed-component) is removed and
replaced with an oru logistics spare *period* :store 6 :nogen)

(defmop m-auto-reconfig (m-processing-mop)

(configuration m-network)

(defun infer (mopname &rest slots)

(setf slots (remove nil slots))

(and *verbose* (st (format nil "~A"

Inferring--> ~S" mopname)))

(activate mopname slots nil t))

(defmop 1-m-system-sm (m-sm) instance)

(add-assoc-fn m-auto-reconfig m-auto-reconfig-fn)

(cond

((equal (mop-type mop) 'instance)

(infer ,m-reconfiguration

(list 'configuration (inherit-filler 'configuration mopname))

(defun m-auto-reconfig-fn (mopname)

(let ((mop (name->mop mopname)))

(equal (mop-type mop) 'instance)

(infer ,m-reconfiguration

(list 'configuration (inherit-filler 'configuration mopname)))

(defphrase m-auto-reconfig (configuration) reconfiguration is effected

automatically *period* :store 2 :nogen)

(defphrase m-bypass-node (result) in a reconfigured state with
the failed (failed-component) bypassed :store 4 :nogen)

(defphrase m-operational the (object) remains in operation :store 3 :nogen)

; Unification mops collect together procedural steps and glom them
; together into the one big procedure that they represent. Different
; procedural segments may end up contributing different slots to the
; overall procedural mop that is created.

(defmop m-unify-mops (m-processing-mop)

(unify m-root)

(defmop m-unify-correction-mops (m-unify-mops)

(unify m-failure-correction))

(defpat m-unify-correction-mops unify :activator m-context :mode case)

(add-assoc-fn m-unify-mops m-unify-correction-mops-fn)

; This assoc fn creates a unified correction procedure given
; subcomponents of that procedure.

; Unification is a necessary step, given that parts of a
; procedure are distributed in different ways within a text. For
; example, one part may be a pointer to the type of correction plan

; that is being used, while another may represent actual slots in that

; procedure. The problem with this implementation is that it

; is not generalized for procedures in general. This function should

; be merged with other unification functions to create a rationality

; generalized unification procedure.

(defun m-unify-correction-mops-fn (mopname)

(let ((mop (name->mop mopname)))

(cond

((equal (mop-type mop) 'instance)

(let* ((unifylist (for (slot :in (mop-slots mop))

(:save (caar slot)))

(absts

(for (elem :in unifylist)

:when elem

(:length (for (elem :in unifylist)

:when elem

(:abst nil)

(slots nil))

(cond

((> (length (for (elem :in unifylist)

:when elem

(:splice (let (temp (mop-slots (name->mop elem)))

(for (ichor :in (am-owners (car (mop-ams

(name->mop elem))))

:when (let ((l1mop (name->mop ichor)))

(and (instance-mopp l1mop

(abstp (name->mop

'm-hl-pattern) l1mop)))

(:setf slots (for (elem :in unifylist)

:when elem

(:hpips

(for (h1lips :in hlips)

:do (let ((hl1mop (name->mop (car

(mop-absts (name->mop hl1p))))))

(setf (mop-specs hl1mop)

(remove hlip (mop-specs hl1mop)))

(setf (mop-slots mop) (remove elem

(mop-slots mop))

temp))

(setf abst (car absts))


```

(first (am-parent am)))
(replace-slot steps :sequence-of-steps result)))))

(defpharse m-bypass-node The conditions associated with the replacement
  of a RC within the network configuration are such that the network
  is already in a reconfigured state supplying full services *period*.
; store 2 :mogen)

(defmop m-disambiguate-none (m-processing-mop)
  (add-assoc-fn m-disambiguate-none none m-none-fn)
  (dephrase m-disambiguate-none none *period*)
  (add-assoc-fn m-disambiguate-none none m-none-fn)
  (cond
    ((and (instance-mopp mop)
          (contextp 'm-failure-effect))
     (activate 'i-m-null-procedure nil
              (make-am :start (car (mop-ams mop)))
              :finish (am-finish (car (mop-ams mop)))))
      t)))
  (dephrase m-failure-effect (effect ed-component) *colon*
  (sequence-of-steps m-procedure) :nogen)
  (let ((mop (name->mop m pathname)))
    (cond
      ((and (instance-mopp mop)
            (contextp 'm-failure-effect))
       (activate 'i-m-null-procedure nil
              (make-am :start (car (mop-ams mop)))
              :finish (am-finish (car (mop-ams mop)))))
         t)))
    ))))

;;;

;;; ENTITIES
;;;

;;; EVENTS
;;;

;;; GROUPS
;;;

;;; MODES
;;;

;;; PROCEDURES
;;;

(defmop m-build-procedure (m-processing-mop)
  (step m-event-step))
(defpat m-build-procedure step :activator m-context :mode case)
(add-assoc-fn m-build-procedure m-build-proc-fn)

; This function builds a procedure from a sequence of steps. The
; basic idea is to take each event that occurs in the procedure, look
; up the set of procedures that that event occurs in, and intersect
; all such sets for each step. Hopefully that intersection will be the
; one procedure that best characterizes the given step. If there is
; more than one procedural candidate, a warning is printed -- since
; this doesn't occur in the first case text, I haven't elaborated
; strategies of dealing with this event any further than that.

; This procedure takes the intersection of the lists of backpointers
; given to it.
(defun filler-intersection (list1 list2 &aux result)
  (remove-duplicates
   (for (elem :in list1)
        (when (or (setf result nil)
                  (setf result
                        (intersection list1 list2)))))))

; The next two functions get a list of backpointers within various
; scopes for use in tracing which procedures and event occurs in.
(defun get-immediate-back-ptrs (elem &aux result)
  (setf result (for (abst :in (mop-all-absbs (name->mop elem)))
                  :when (not (equal abst 'm-root))
                  :splice (mop-back (name->mop abst)))))
  (for (item :in result)
       :save (append item (list elem)))))

(defun get-all-back-ptrs (elem &aux result)
  (setf result (for (abst :in (mop-all-absbs (name->mop elem)))
                  :when (not (equal abst 'm-root))
                  :splice (mop-back (name->mop abst)))))
  (for (item :in result)
       :save (append item (list elem)))))


```

```

(member-if #'(lambda (x) (equal (cadr elem) (cadr x)))
  :splice (list (car result) elem)))

; Utility function to remove-duplicates using structural features
; rather than pointer features for lists.

(defun remove-duplicates (l)
  (let ((newl nil))
    (for (elem :in l)
      :when (not (lmember elem newl))
      :save (setf newl (cons elem newl)))
    newl))

; A filter designed to prevent some mops from getting
; considered in some processing stage. The fact that this filter is
; necessary suggests that the approach in which this procedure appears
; needs some refinement, or that the representation hierarchies need
; to be changed somewhat.

(defun crude-filter (l)
  (for (elem :in l)
    :when (cond
            ((equal (car elem) 'sequence-of-steps) t)
            ((not (numberp (car elem))) nil)
            ((equal (cadr elem) 'm-and-group) nil)
            ((equal (cadr elem) 'm-or-group) nil)
            ((equal (cadr elem) 't) t)
            ((equal (cadr elem) 't) t)
            :else nil)
    :save elem))

;; STATES
;; SYSTEMS

```

```

*****  

*****  

***** File: question.lsp  

*****  

***** FANSYS: A Computer Model of Text Comprehension and Question  

***** Answering for Failure Analysis  

*****  

***** Developed by: Sergio J. Alvarado  

***** Ronald K. Braun  

***** Kenrick J. Mock  

*****  

***** Artificial Intelligence Laboratory  

***** Computer Science Department  

***** University of California  

***** Davis, CA 95616  

*****  

***** Funds for the support of this study have been allocated by the  

***** NASA-Ames Research Center, Moffett Field, California, under  

***** Interchange No. NCA2-721.  

*****  

*****  

***** Question/Answering Module  

*****  

***** The code within this section deals with the process of finding  

***** and returning cases for question answering. The mocknops.lsp  

***** code must have been previously loaded before this can run,  

***** as there is great deal of interaction and sharing of functions.  

*****  

***** Global Variable to determine if we should be wordy or not  

***** In verbose mode, extra debug information is given  

(setf *verbose* 't)  

*****  

***** FindHeader  

***** Given some MOP and an item that will be a filler for a slot, such as  

***** EntryMop = M-CASE and item=MRC, find the header for which the item  

***** is a filler. So if M-CASE has : (Failed-component M-ORU), the M-RC  

***** is a spec of M-ORU so we have a match and return (failed-component m-rc).  

***** Used to create a search query.  

*****  

(defun FindHeader (entrymop item)
  (let ((mp (name->mop entrymop))
        (childmop (name->mop item)))
    (for (i :in (get-all-slots mp))
         :WHEN (and (CheckValidity (cadr i))
                    (MyAbstp (cadr i) childmop))
         :FIRST (list (car i) item)))  

(defun CheckValidity (mopfiller)
  (cond ((equal mopfiller 'M-ROOT)
         (cond (*verbose* "Filler=M-ROOT! Potential Rep Error. Ignoring now...~*"))
               (st (format nil "Format nil ~A~*"))
               nil)
        (t ,t)))  

***** AddBuildSlots  

***** BuildSlots calls FindHeader for each item in the given itemlist  

***** so that we can build up a slotlist query for all of the known  

***** information.  

*****  

(defun BuildSlots (entrymop itemlist)
  (let ((newslots nil)
        (for (i :in itemlist)
             :DO
               (cond
                 ((listp i)
                  (setf newslots (cons (FindHeader entrymop i) newslots)))
                 (newslots)))
  

***** AddWildCards  

***** For the Entry Mops, do a : (mopfiller *) as the search query  

***** Since these are the things we are looking for  

***** This allows us to traverse indices which match the requested  

***** item (and is obviously unknown if it is requested).  

*****  

(defun AddWildCards (slotpairs moplist entrymop)
  (let ((newslots nil)
        (for (i :in moplist)
             :DO
               (setf newslots (cons
                               (list (car (FindHeader entrymop i))
                                     newslots)))
  

               (append slotpairs newslots)))  

***** AddEntrySlots  

***** For the Entry Mop, do a : (equal mopfiller *) when mopfiller is  

***** unspecified in the query and a member of the entry mop, so we  

***** can elaborate on it. Note that this returns a built-up query.  

***** This is only called when we're doing elaboration.  

*****  

(defun AddEntrySlots (entry slots)
  (let ((new nil) (junk nil)
        (for (i :in (ExtractFillers entry))
             :DO
               (cond
                 ((setf junk (car-member i slots))
                  (setf new (append new (list (cons 'equal junk))))))
                 (t
                  (setf new (append new (list (cons 'equal (list i '*))))))
  

                  (cons 'or new)))
  

***** ExtractFillers  

***** Extracts the Fillers of all slots of a mop. So if the slots are:  


```

```

;; (failed-component X) (height Y) (color Z) return the list
;; (failed-component height color)
;;
;; ExtractFillers (mopname)
;;   (for (i :in (get-all-slots (name->mop mopname)))
;;     :SAVE (car i))

;*****+
;*** UserAbst
;*** If its a slotless mop, use the abstraction instead. Otherwise
;*** If the given mop... little hack here to do mop comparisons and
;*** use the search query.
;*** Such. Used for the search query.
;*** Used for the search query.

(defun ExtractFillers (mopname)
  (for (i :in (get-all-slots (name->mop mopname)))
    :SAVE (car i))

;*****+
;*** AnswerHeader
;*** In some cases the answer we want is the slot, in others it is the
;*** entire MOP retrieved. When it's the slot, the following functions
;*** (starting from AnswersSlots) takes the list of found mops and the
;*** requested mop list and returns the slot roles which match those given
;*** in the request list.
;*** In the request list.

;*****+
;*** UserAbst (itemlist)
;*** (for (i :in itemlist)
;***   :SAVE
;***   (cond
;***     ((listp i)
;***      (cond
;***        ((null (mop-slots (name->mop (cadr i))))
;***         (list (car i) (car (mop-absts (name->mop (cadr i))))))
;***        ((t i))
;***          (null (mop-slots (name->mop i)))
;***            (car (mop-absts (name->mop i)))
;***              (t i)))
;***        (t i)))))

;*****+
;*** UserAbst2
;*** New Version 1/26/93 to fix bugs. Old version of checking the next
;*** mop over as the requested wasn't working properly in all cases,
;*** in particular when the requested item is an ORU.
;*** In particular when the requested item is an ORU.

(defun UserAbst (itemlist)
  (for (i :in itemlist)
    :SAVE (cadr i)))

;*****+
;*** Multiple-Find (instmop moplist)
;*** (let ((temp nil)
;***       (for (i :in moplist)
;***         (AnswerHeader i instmop))
;***       :WHEN (setf temp (AnswerHeader i instmop))
;***       (SPICE temp)))
;***       (for (i :in (get-all-slots mp))
;***         (let ((mp (name->mop instmop)))
;***           (for (i :in (get-all-slots mp))
;***             (when (MyAbtp absmop (name->mop (cadr i)))
;***               :SAVE (cadr i)))))

;*****+
;*** Multiple-Find (instmop moplist)
;*** (let ((temp nil)
;***       (for (i :in moplist)
;***         (AnswerHeader i instmop))
;***       :WHEN (setf temp (AnswerHeader i instmop))
;***       (SPICE temp)))
;***       (for (i :in (get-all-slots mp))
;***         (let ((mp (name->mop instmop)))
;***           (for (i :in (get-all-slots mp))
;***             (when (MyAbtp absmop (name->mop (cadr i)))
;***               :SAVE (cadr i)))))

;*****+
;*** GetExplodedMop
;*** Try retrieving exploded MOP from hash table first. If it fails
;*** then manually explode the MOP, then store it in the hash table
;*** for future reference.
;*** Hash table retrieval is much faster than re-exploding a MOP.
;*** Once we have exploded a MOP, be sure to store it in the hash
;*** table to facilitate future retrieval.

;*****+
;*** GetExplodedMop (mopname)
;*** (let ((temp nil))
;***       (setf temp (gethash mopname *Exploded*)))
;***       (cond
;***         ((null temp)
;***           (setf temp (reformat (explode-it mopname)))
;***             (setf (gethash mopname *Exploded*) temp)))
;***         (temp))

;*****+
;*** Given a list of slots, explode the RHS of each slot.

;*****+
;*** ExpandSlots (slotpairs)
;*** (defun ExpandSlots (slotpairs)
;***   (let ((newslots nil)
;***         (for (i :in slotpairs)
;***           (do

;*****+
;*** UserAbst3 (mopname)
;*** (cond
;***   ((equal (mop-type (name->mop mopname)) 'MOP)
;***     mopname)
;***   (t
;***     :DO

```

```

(let ((equiv nil)
      (setf equiv (mop-absts (name->mop mopname)))
      :WHEN (SuperAbst i equiv)
      :FIRST 1)))
*****  

;; Main Question/Answer routine. Then it determines if the slots are to
;; be returned, or if the whole MOP should be returned - this depends on
;; what entry point was picked. If we had to go up the backpointers, then
;; it'll be a slot. If it's the starting point it will be the entire case.
;; Build the query (has to search the slots "backwards" to find the matching
;; filler to use for search) and do a direct search. If that fails, then
;; add (requested *) to the query and try again. If that fails, then add
;; (all *) to the query, where "all" are unspecified slots in the query.
;; This means that a "last resort" is to basically do a depth-first
;; search on everything in memory.
*****  

(not
  (for (l1 :in moplist)
    :WHEN (not (myabstp topmop (name->mop 1)))
    :FIRST 't))
*****  

;; BuildsearchList
*****  

;; Given an entry mop where it doesn't directly index instances, go
;; down the hierarchy until we reach the ones that do index instances
;; and use those as the entry point(s).
;; Works as long as all specs are either instances or mops; no combo
;; of the two!
*****  

(defun BuildSearchList (entrymop)
  (let ((newlist nil) (specs nil))
    (loop
      (setf newlist (append newlist (BuildSearchList 1))))
      (setf specs (mop-specs (name->mop entrymop)))
      (cond
        (specs
          (cond
            (equal (mop-type (name->mop (car specs))) 'INSTANCE)
              (setf newlist (list entrymop)))
            (t
              (for (l1 :in specs)
                :DO
                  (setf newlist (append newlist (BuildSearchList 1)))))))
        (My-Remove-Dupe newlist)))
*****  

;; Strippackageinfo
*****  

;; This strips off stuff from input with package info.
;; For example, if QA is called from the KR package, the variables
;; look like "kr::failure-mode" etc. This causes problems, so
;; strip off the KR using this function.
*****  

(defun StripPackageInfo (l)
  (for (l1 :in l)
    :SAVE
    (cond
      ((eql l 'temp) (StripPackageInfo l))
      (t
        (princ "Found good match. Elaborate anyway? (Y/n) ")
        (setf temp (read()))
        (cond ((equal temp 'N) (setf temp nil) (setf abort 't))
              (t (setf temp 't)))))))
*****  

;; environments
*****  


```

```

        (for (i :in searchlist)
      ))
        (:DO
          (cond
            ((setf temp (DirectSearch i query mpcent))
             (setf foundlist (append temp foundlist))))
            (cond
              (*verbose*
                (st (format nil "Suggested cases are: ~S~&" foundlist)
                    )
                  (format t "~-&~&Suggested cases are: ~A~&" )
                  (format t "Suggested cases are: ~A~&" )
                  (SpeakIt returnitem foundlist moplist)
                )
              )
              ((my-remove-dupe
                (cond
                  (equal returnitem 'SLOT)
                  (AnswerSlots foundlist moplist)
                  (t foundlist)))
                )
              )
            )
          )
        )
      )
    )
  )
  (cond
    (*verbose*
      (st (format nil "Elaborating using requested information.~&"))
        (st (format nil "Elaborating using anything. Indices: ~S~& origslots")
          )
          (setf query (pairs-query slots))
          (for (i :in searchlist)
            (:DO
              (cond
                ((setf temp (DirectSearch i query 1))
                 (setf foundlist (append temp foundlist)))
                  (format t "~-&Results of elaboration: ~&")
                  (and *verbose* (st (format nil "~-&Results of elaboration: ~&"))
                      (cond (*verbose* (st (format nil "~-&~-&~-&"))
                        (SpeakIt returnitem foundlist moplist)
                      )
                      (cond
                        (null foundlist) (setf temp 't)
                        (null abort)
                        ((princ "Found good match. Elaborate further anyway? (Y/n) ")
                         (if (read) (setf temp nil) (setf temp 't)))
                        (cond ((equal temp 'N) (setf temp nil) (setf abort 't))
                          (t (setf temp 't)))
                        (cond
                          (temp
                            (format t "Elaborating on unspecified entry slots...~&")
                            (cond
                              (*verbose*
                                (st (format nil "Elaborating on unspecified entry slots...~&"))
                                (st (format nil "Read() Indices: ~S~&" (match-attribute-list
                                  (AddEntrySlots entry origslots)))
                                  (AddEntrySlots entry slots)))
                              )
                            )
                            (setf query (AddEntrySlots entry slots))
                            (land *verbose*)
                            (for (i :in searchlist)
                              (:DO
                                (cond
                                  ((setf temp (DirectSearch i query 1))
                                   (format t "~-&Results of secondary elaboration: ~&"))
                                  (and *verbose*
                                    (st (format nil "~-&Results of secondary elaboration: ~&"))
                                    (cond (*verbose* (st (format nil "~-&~-&~-&"))
                                      (SpeakIt returnitem foundlist moplist)
                                    )
                                    (cond
                                      (null abort)
                                      ((princ "Elaborate using partial match? (Y/n) ")
                                       (if (read) (setf temp nil)
                                         (cond ((equal temp 'N) (setf temp nil)
                                           (t (setf temp 't))
                                           (princ "Enter & match desired (0..1) : ")
                                           (setf mpcent (read())))
                                         )))
                                      (cond
                                        (item
                                          (format t "~-&Elaborating on Partial Match...~&")
                                          (cond
                                            (*verbose*
                                              (st (format nil "Elaborating on Partial Match...~&"))
                                              (st (format nil "Indices : ~S~& origslots"))
                                            )
                                          )
                                        )
                                      )
                                    )
                                  )
                                )
                              )
                            )
                          )
                        )
                      )
                    )
                  )
                )
              )
            )
          )
        )
      )
    )
  )
  (cond
    (*verbose*
      (st (format nil "Variant of MOP-Search. This does a direct-search using the given
        attribute-reqr list, and doesn't do elaboration and other such stuff.
        Elaborations are done at a higher level - this function is passed
        a new attribute-reqr and match percent parameters which are less
        constrained, giving more information back.
      ))
    )
  )

```

```

;; See the MOP-Search documentation for details.
*****  

(defun DirectSearch (sourcemap attribute-reqr mpcent)
  (let ((found nil)
        (bestval nil)
        (best nil))
    (cond
      (*verbose*
       (st (format nil "~>Performing Search. Entry MOP->S-&~" sourcemap)))
      (setf found (MOP-Search1 sourcemap attribute-reqr))
      (setf found (remove-duplicates found))
      (cond
        (*verbose*
         (st (format nil "Found: ~S-&~ found")))
        (setf found (all-match found attribute-reqr))
        (cond
          (*verbose*
           (st (format nil "Values: ~S-&~ found"))
           (st (format nil "Picking out those that match ~S-&~ (* mpcent 100)"))
           (for (i :in found)
                :WHEN (>= (cadr i) mpcent)
                :SAVE (car i))))
        )
      )
    )
  )
*****  

;; All-Match
;; Given a list of mops and a query, figure out how well each one
;; matches and return it in a list like: (mop1 val1) (mop2 val2) ...
;; The val's will be Percentages of match as determined by the
;; matchpercent function.
*****  

(defun all-match (moplist query)
  (let ((mval nil)
        (caseslots nil)
        (newlist (match-attribute-list query)))
    (for (i :in moplist)
      :DO
      (cond
        (equal 'MOP (mop-type (name->mop 1)))
        (setf mval (append mval (list '0)))
        (equal 'INSTANCE (mop-type (name->mop 1)))
        (setf caseslots (reformat (explode-it 1)))
        (setf caseslots (GetExplodedMop 1))
        (setf mval (append mval
                           (list (MatchPercent newlist caseslots))))))
    )
  )
*****  

;; MergeLists
;; merges two lists together
*****  

(defun MergeLists (l1 l2)
  (cond
    ((null l1) nil)
    (t (cons (list (car l1) (car l2)) (MergeLists (cdr l1) (cdr l2))))))
*****  


```

```

;;;; MyPrettyPrint
;;;; Format a list nicely with word wrap.  Deals with list of symbols
;;;; or list of strings.
*****  

(defun MyPrettyPrint (l)
  (let ((x 0))
    (for (i :in l)
      :DO
      (progn
        (cond
          ((symbolp i)
           (cond ((> x 45)
                  (setf x 4)
                  (terpri) (princ " ")
                  (t (setf x (+ x (length (format nil "~S~ 1)))))))
              (format t "~S~ 1")
              (stringp 1)
              (cond ((> x 45)
                     (setf x 4)
                     (terpri) (princ " ")
                     (t (setf x (+ x (length 1))))))
              (princ 1) (princ " ")))
        ,t)
      )
    )
*****  


```

the ring concentrator *q-mark*)

```
*****  
** FANSY: A Computer Model of Text Comprehension and Question  
** Answering for Failure Analysis  
**  
** File: qa-phrases  
** Developed by: Sergio J. Alvarado  
**           Ronald K. Braun  
**           Kenrick J. Mock  
**  
**           Artificial Intelligence Laboratory  
**           Computer Science Department  
**           University of California  
**           Davis, CA 95616  
**  
** Funds for the support of this study have been allocated by the  
** NASA-Ames Research Center, Moffett Field, California, under  
** Interchange No. NCA-721.  
*****  
;  
; This file organizes the phrases, mops, and functions necessary for  
; the question answering aspect of the parser.  
*****  
;  
; Set the mode to question answering.  
  
(setf *mode* 'qa)  
;  
; These are sample questions that may be used to query the case base.  
;  
; Type I Questions  
  
(setf t1ql1 '(what is the failure mode for the ring concentrator *q-mark*))  
(setf t1ql2 '(what is the failure cause for the ring concentrator when the  
failure mode is loss of output - failure to start *q-mark*))  
(setf t1q3 '(what is the failure detection procedure for the ring  
concentrator when the failure mode is loss of output - failure to  
start *q-mark*))  
(setf t1q4 '(what is the failure correction procedure for the ring  
concentrator when the failure mode is loss of output - failure to  
start *q-mark*))  
(setf t1q5 '(what is the failure effect for the ring concentrator when  
the failure mode is loss of output - failure to start *q-mark*))  
(setf t1q6 '(what is the failure detection procedure when the failed  
component is the ring concentrator *q-mark*))  
;  
; Type II Questions  
  
(setf t2q1 '(what is the failure cause for the ring concentrator *q-mark*))  
(setf t2q2 '(what is the failure detection procedure when the failed  
component is the ring concentrator *q-mark*))  
(setf t2q3 '(what is the failure correction procedure when the failed  
component is the ring concentrator *q-mark*))  
(setf t2q4 '(what is the failure effect when the failed component is  
the ring concentrator *q-mark*))  
;  
; Type III Questions  
  
(setf t3ql1 '(what are the detection steps in the next node detection  
procedure when the failed component is the ring concentrator *q-mark*))  
(setf t3q2 '(what are the correction steps in the bypass node  
procedure when the failed component is the ring concentrator *q-mark*))  
;  
; The mops that characterizes a question instance. Two different  
; question mops are used, since one has a slightly  
; different processing strategy based on the requested slot.  
  
(defmop m-question (m-processing-mop)  
  (state m-state-assertion)  
  (requested m-question-constraint)  
  (given m-question-constraint))  
  
(defmop m-question2 (m-processing-mop)  
  (state m-state-assertion)  
  (requested m-and-group)  
  (given m-question-constraint))  
;  
; Hard code in a link in the abst hierarchy.  
  
(setf (mop-absbs (name->mop 'm-group)) (cons 'm-question-constraint  
  (mop-absbs (name->mop 'm-group))))  
;  
; add-assoc-fn m-question m-question-fn)  
(add-assoc-fn m-question m-question-fn)  
;  
; The question phrases. These are pretty formal and rigid in  
; definition, thereby severely restricting the manner in which  
; questions can be phrased. The addition of more elaborate patterns  
; would alleviate this problem.  
;  
(defphrase m-question2 what are the (requested) in the (given) when the  
  (state) *q-mark*)  
;  
(defphrase m-question what is the (requested) and the (requested) for  
  the (given) when the (state) and the (state) *q-mark*)  
:opt (5 6 7 8 9 10 11 12 13 14 15 16)  
;  
; Misc. phrases.  
;  
(defphrase m-detection-and-group detection steps :nogen)  
(defphrase m-correction-and-group correction steps :nogen)
```

```

(defphrase m-bypass-node network self-reconfiguration :store 2
  :nogen)
(defphrase m-replace-with-spare replace with spare :nogen)
(defphrase m-switch-or-bypass switch or bypass via backup
  :nogen)
(defphrase m-bypass-node bypass the failed node :nogen)

; The basic question function, which extracts the requested and given
; slots from the question mop, and calls Ken's hierarchy with them.

(define m-question-fn (mopname)
  (let ((mop (name->mop mopname)))
    (cond
      ((instance-mopp mop)
       (let ((given (for (slot :in (mop-slots mop))
                         :when (equal (car slot) 'given)
                         :then (cadr slot)))
            (requested (for (slot :in (mop-slots mop))
                           :when (equal (car slot) 'requested)
                           :then (cadr slot)))
            (states (for (slot :in (mop-slots mop))
                         :when (equal (car slot) 'state)
                         :then (cadr slot)))
            (results nil))
         (for (state :in states)
              :do (setf given (cons (list (determine-slot-name
                                           (role-filler 'name state))
                                         (role-filler 'filler state))
                                     (tqa given requested))))))
      (t
       (error "Unknown question type ~S" mopname)))))

; This function converts the mopname given into a suitable slot name.

; Ken's stuff should determine the slot name by taking
; the entry point and the given mopname, and looking through the slots
; of the entry point for a slot with the filler matching mopname.

(define determine-slot-name (mopname)
  (let ((mop (name->mop mopname)))
    (cond
      ((member 'm-failure-mode (mop-all-absts mop))
       (mode)
       (member 'm-failure-cause (mop-all-absts mop)))
      ((member 'm-failure-detection (mop-all-absts mop))
       ('cause)
       (member 'm-failure-detection (mop-all-absts mop)))
      ('detection)
      ((member 'm-failure-correction (mop-all-absts mop))
       ('correction)
       (member 'm-failure-effect (mop-all-absts mop)))
      ('effect)
      ((member 'm-oru (mop-all-absts mop))
       ('failed-component))))))

(defphrase m-state-assertion (name) is the (filler) :store 2 :opt (3)
  (defmop *q-mark* (m-hard-punctuation) instance)
  (defmop m-no-effect (m-processing-mop) )
  (defphrase m-no-effect there is no failure effect :store 3)
  (add-assoc-in m-no-effect m-no-effect-fn)

; If there is no effect on the mission component, we must infer that
; the null procedure is being used, since each effect expects a
; procedure.

(define m-no-effect-fn (mopname)
  (let ((mop (name->mop mopname)))
    (result nil)
    (cond
      ((instance-mopp mop)
       (setf result (infer 'm-failure-effect
                           'sequence-of-steps 1-m-null-procedure)))
      (activate 'm-state-assertion (list 'name m-failure-effect)
                (list 'filler result)
                (make-aim :start (am-start (car (mop-ams mop))) t)))
      (ifinish (amfinish (car (mop-ams mop))) given)))
    (t
     (error "Unknown question type ~S" mopname)))))

(defphrase m-failure-correction failure correction procedure :store 2
  :nogen)
(defphrase m-failure-correction correction procedure :nogen)
(defphrase m-failure-detection failure detection procedure :store 2
  :nogen)
(defphrase m-failure-detection detection procedure :nogen)
(defphrase m-failure-effect failure effect :store 2 :nogen)
(defphrase m-failure-cause failure cause :store 2 :nogen)
(defphrase m-operational-erroneous-output operational - erroneous
  output :nogen)

(defphrase m-oru failed component :store 2 :nogen)
(defphrase m-failure-mode failure mode :store 2 :nogen)
(defphrase m-next-node-detection next node detection :store 2
  :nogen)
(defphrase m-post-detection power-on self test :store 3
  :nogen)
(defphrase m-ecc-detection error correction code detection
  :store 2 :nogen)

(defphrase m-heartbeat-detection system management heartbeat messages
  :store 3 :nogen :opt (1 2))

(defphrase m-bypass-node bypass node procedure :nogen)
(defphrase m-non-operational-replacement check connections or replace
  :nogen)
(defphrase m-lookup-backup automatic backup from lookup-table
  :nogen)

```



```

;;;; Parameters and stuff.

;;;; CLX specials

(defvar *display* nil)
(defvar *screen* nil)
(defvar *root* nil)
(defvar *gcontext* nil)
(defvar *black-pixel* nil)
(defvar *white-pixel* nil)

;;;; Inspect-Length is the number of components that will be displayed in a
;;;; window at any one time. If an object has more than Inspect-Length
;;;; components, we generally put it in a scrolling window. Inspect-Level
;;;; might someday correspond to Print-Level, controlling the amount of
;;;; detail and mouse-sensitivity we get inside components, but for now
;;;; it's ignored.

(defparameter inspect-length 10)
(defparameter inspect-level 1)

;;;; Inspect-Print-Level and Inspect-Print-Length are used by IPrintn-To-String
;;;; to generate the textual representation of components.

(defparameter inspect-print-length 10)
(defparameter inspect-print-level 3)

(defun iprintn-to-string (object)
  (let ((*print-length* inspect-print-length)
        (*print-level* inspect-print-level)
        (*print-pretty* nil))
    (prin1-to-string object)))

;;;; Inspect-Line-Length is a hack used in only one place that we should get
;;;; rid of someday.

(defparameter inspect-line-length 80)

;;;; We use font structures to keep stuff like the character height and width
;;;; of a font around for quick and easy size calculations. For variable width
;;;; fonts, the width slot will be Nil.

(destruct (font (:constructor make-font (name font height ascent width)))
  name
  font
  height
  ascent
  width)

;;;; The *Header-Font* is a big font usually used for displaying stuff in
;;;; the header portion of an object display. *Entry-Font* is used as the
;;;; main "body font" for an object, and *Italic-Font* is used for special
;;;; stuff.

(defparameter header-font-name "*-courier-bold-r-normal---*-240-*")
(defvar *header-font*)

(defparameter entry-font-name "*-courier-bold-r-normal---*-180-*")
(defvar *entry-font*)

(defparameter italic-font-name "*-courier-bold-o-normal---*-180-*")
(defvar *italic-font*)

(defparameter cursor-name "inspect1.cursor")
(defvar *cursor*)

(defparameter cursor-d-name "library:inspect1-d.cursor")
(defvar *cursor-d*)

(defparameter cursor-s-name "library:inspect1-s.cursor")
(defvar *cursor-s*)

;;;; This file contains the help message for the inspector. The text in the
;;;; file must not extend past the 72nd column, and any initial whitespace on
;;;; a line must be built on the space character only. The window that displays
;;;; this text is too small in height for easy reading of this text.

(defparameter help-file-pathname "memory/myinspect.help")

```

```

(black (xlib:make-color :red 0.0 :green 0.0 :blue 0.0)
(cursor (xlib:create-cursor :source cursor-pixmap :mask mask-pixmap
                           ;x cursor-x-hot :y cursor-y-hot
                           ;foreground black :background white)))
(xlib:free-pixmap mask-pixmap)
(xlib:free-pixmap cursor-pixmap)
(cursor))

(defun bitvec-list-to-pixmap (bvl width height)
  (let* ((image (apply #'xlib:bitmap-image bv1))
         (pixmap (xlib:create-pixmap :width width :height height
                                      :drawable *root*))
         (gc (xlib:create-qcontext :drawable pixmap
                                   :function boole-1
                                   :foreground "black-pixel"))
         (xlib:put-image pixmap gc image :x 0 :y 0 :width 16 :height 16 :bitmap-p t)
  (xlib:re-gcontext gc)
  (pixmap))

(defun invert-pixmap (pixmap)
  (let* ((width (xlib:drawable-width pixmap))
         (height (xlib:drawable-height pixmap))
         (inv-pixmap (xlib:create-pixmap :width width :height height
                                         :drawable *root*
                                         :depth (xlib:screen-root-depth *screen*)))
         (gc (xlib:create-qcontext :drawable inv-pixmap
                                   :function boole-1
                                   :foreground "black-pixel"
                                   :background "white-pixel"))
         (xlib:copy-area pixmap gc 0 0 width height inv-pixmap 0 0)
  (xlib:free-qcontext gc)
  (inv-pixmap))

;;; Font and cursor support

(defun open-font (name)
  (let* ((font (xlib:open-font *display* name))
         (max-width (xlib:max-char-width font))
         (min-width (xlib:min-char-width font))
         (width (if (= max-width min-width) max-width nil))
         (ascent (xlib:max-char-ascent font))
         (height (+ (xlib:max-char-descent font) ascent)))
    (make-font name font height ascent width)))

(defun get-cursor-pixmap-from-file (name)
  (let* ((pathname (probe-file name)))
    (if pathname
        (let* ((image (xlib:read-bitmap-file pathname))
               (pixmap (xlib:create-pixmap :width 16 :height 16
                                         :depth 1 :drawable *root*))
               (gc (xlib:create-qcontext :drawable pixmap
                                         :function boole-1
                                         :foreground "black-pixel"
                                         :background "white-pixel")))
          (xlib:put-image pixmap gc image :x 0 :y 0 :width 16 :height 16)
        (xlib:free-qcontext gc)
        (values pixmap (xlib:image-x-hot image) (xlib:image-y-hot image)))
    (values nil nil nil)))

(defun open-cursor (name)
  (multiple-value-bind
      (cursor-pixmap cursor-x-hot cursor-y-hot)
      (get-cursor-pixmap-from-file name)
    (declare (ignore cursor-x-hot mask-y-hot))
    (let* ((white (xlib:make-color :red 1.0 :green 1.0 :blue 1.0))
           (mask-pixmap mask-x-hot mask-y-hot)
           (get-cursor-pixmap-from-file (make-pathname :type "mask" :defaults name)))
      (values mask-pixmap cursor-x-hot cursor-y-hot)
      (get-cursor-pixmap-from-file name)
      (gat-cursor-pixmap-from-file name)
      (multiple-value-bind
          (multiple-value-bind
              (cursor-pixmap cursor-x-not mask-y-not)
              (get-cursor-pixmap-from-file name)
            (declare (ignore cursor-x-not mask-y-not))
            (let* ((white (xlib:make-color :red 1.0 :green 1.0 :blue 1.0)))
              (values mask-pixmap cursor-x-not cursor-y-not)
              (get-cursor-pixmap-from-file (make-pathname :type "mask" :defaults name)))))))

```

```

;;;; Inspect-Init
;;;; Inspect-Init sets all this stuff up, using *Inspect-Initialized* to
;;;; know when it's already been done.

(defvar *inspect-initialized* nil)

(defun inspect-init ()
  (unless *inspect-initialized*
    (multiple-value-setq (*display* *screen*) (ext:open-clx-display)
      (ext:carefully-add-font-paths
        "display"
        (mapcar #'(lambda (x)
          (concatenate 'string (namestring x) "fonts/"))
          (search-list "library:")
        )
        (setq *root* (xlib:screen-root "screen"))
        (setq *black-pixel* (xlib:screen-black-pixel "screen"))
        (setq *white-pixel* (xlib:screen-white-pixel "screen"))
        (setq *gcontext* (xlib:create-gcontext :drawable *root* :function boole-1
          :foreground *black-pixel*
          :background *white-pixel*))
        (setq *cursor* (open-cursor cursor-name))
        (setq *cursor-d* (open-cursor cursor-d-name))
        (setq *cursor-s* (open-cursor cursor-s-name))
        (setq *header-font* (open-font header-font-name))
        (setq *entry-font* (open-font entry-font-name))
        (setq *italic-font* (open-font italic-font-name))
        (setq *up-arrow* (bitvec-list-to-pixmap arrow-bits 16 16))
        (setq *up-arrow-i* (invert-pixmap *up-arrow*))
        (setq *down-arrow* (bitvec-list-to-pixmap (reverse arrow-bits) 16 16))
        (setq *down-arrow-i* (invert-pixmap *down-arrow*))
        (text:enable-clx-event-handling "display" 'inspector-event-handler)
        (setq *inspect-initialized* t)
      )
    )
  )

|---- For debugging...
|---- (defun inspect-reinit (optional (host "unix:0.0"))
|----   (let ((win nil)
|----     (setq *inspect-initialized* nil)
|----     (when *display*
|----       (ext: disable-clx-event-handling *display*)
|----       (xlib:close-display *display*)))
|----     (unwind-protect
|----       (progn
|----         (multiple-value-setq
|----           (*display* *screen*)
|----           (text:open-clx-display host))
|----         (setf (xlib:display-after-function *display*)
|----               (xlib:display-finish-output))
|----         (setq *root* (xlib:screen-root "screen"))
|----         (setq *black-pixel* (xlib:screen-black-pixel "screen"))
|----         (setq *white-pixel* (xlib:screen-white-pixel "screen"))
|----         (setq *gcontext* (xlib:create-gcontext :drawable *root*
|----           :foreground *black-pixel*
|----           :background *white-pixel*))
|----         (setq *cursor* (open-cursor cursor-name))
|----         (setq *cursor-d* (open-cursor cursor-d-name))
|----         (setq *cursor-s* (open-cursor cursor-s-name))
|----         (setq *header-font* (open-font header-font-name))
|----         (setq *entry-font* (open-font entry-font-name))
|----         (setq *italic-font* (open-font italic-font-name))
|----       )
|----     )
|----   )
|---- )

```

```

;;;
;; More X Stuff

;; We use display-info structures to associate objects with their graphical
;; images (Display-Items, see below), the X windows that they're displayed in,
;; and maybe even a user-supplied name for the whole thing.

(defstruct display-info
  (:constructor make-display-info (name object display-item window))
  name
  object
  display-item
  window
  (stack nil))

;; "display-infos" is a list of all the live displays of objects.

(defvar *display-infos* nil)

;; CLX window to display-info structure mapping.

(defvar *windows-to-displays* (make-hash-table :test #'eq))

(defun add-window-display-info-mapping (window display-info)
  (setf (gethash window *windows-to-displays*) display-info))

(defun delete-window-display-info-mapping (window)
  (remhash window *windows-to-displays*))

(defun map-window-to-display-info (window)
  (multiple-value-bind (display-info found-p)
      (gethash window *windows-to-displays*)
    (unless found-p (error "No such window as ~S in mapping!" window))
    display-info))

;; Tracking-Mode* is a kind of hack used so things know what to do
;; during modify operations. If it's :source, only objects that are really
;; there will be selectable. If it's :destination, objects that aren't
;; necessarily really there (like the values of unbound symbols) will be
;; selectable.

(defvar *tracking-mode* :source)

;; *Mouse-X* and *Mouse-Y* are a good approximation of where the mouse is
;; in the window that the mouse is in.

(defvar *mouse-x* 0)
(defvar *mouse-y* 0)

```

```

;;;; Event Handling

;; We're interested in these events:
(eval-when (compile load eval)
  (defconstant important-xevents
    '(:key-press :button-press :exposure :pointer-motion
      :enter-window :leave-window))

  (defconstant important-xevents-mask
    '(apply #'xlib:make-event-mask important-xevents)))

(defun inspector-event-handler (display)
  (xlib:event-case (display :discard-p t :force-output-p t :timeout 0)
    ((:exposure) (event-window count)
     (when (zerop count)
       (redisplay-item
        (display-info-display-item (map-window-to-display-info event-window)))))

    ((:key-press) (event-window state code)
     (do-command (map-window-to-display-info event-window)
      (ext:translate-key-event display-code state)))
    t)

    ((:button-press :button-release) (event-key event-window state code)
     (do-command (map-window-to-display-info event-window)
      (ext:translate-mouse-key-event code state event-key)))
    t)

    ((:enter-notify :motion-notify) (event-window x y)
     (cond ((xlib:event-listen display)
            ; if there are other things in the queue, blow this event off...
            nil)
           (t
            (setf *mouse-x* x)
            (setf *mouse-y* y)
            (track-mouse (display-info-display-item
                          (map-window-to-display-info event-window)
                          x y))
            t)))
    t)

    ((:leave-notify) (event-window)
     (track-mouse (display-info-display-item
                   (map-window-to-display-info event-window)
                   -1 -1))
     t)

    ((:no-exposure) ())
    ; just ignore this one
    t)

  (t (event-key)
    (unless (eq event-key :motion-notify)
      (format t "Inspector received unexpected event, ~S, received." event-key)))
  t))

;; Some debugging code...

(xlib:event-cond (display :timeout 0 :peek-p t)
  (t (event-key)
    (unless (eq event-key :motion-notify)
      (format t "Event received: ~S~%" event-key)))
  (format t "Event received: ~S~%" event-key)))

(defun discard-event-on-window (display window type)
  (loop
    (unless (xlib:process-event display :timeout 0

```

```
:handler #'(lambda (&key event-window event-type &allow-other-keys/
  &and (eq event-window
    (eq event-window window)))
```

==== Yet more X stuff.

```

;; Disp-String draws a string, trying to constrain it to not run beyond the
;; *x-constraint*. For variable width fonts, we can only guess about the
;; right length...
;;;

(defun disp-string (window x y string disp-font)
  (declare (simple-string string))
  (let ((font (font-font disp-font)))
    (let ((font-width (font-width disp-font))
          (font-height (font-height disp-font)))
      (length (length string))
      (max-width (if (* x-constraint x) max-window-width)
                 (cond (font-fixed-width font
                                         (let ((end (if (< (* length font-width) max-width)
                                         (let ((fixed-width (font-width font))
                                             (if (< (* length font-width) max-width)
                                                 
```

```

;;;;
;; Draw-Bitmap, Draw-Box, and Draw-Block

(defmacro draw-bitmap (window x y pixmap)
  `(xlib:copy-area pixmap *gcontext* 0 0 16 16 window x y))

(defmacro draw-box (window x1 y1 x2 y2)
  `(xlib:draw-rectangle window *gcontext* x1 y1 (- x2 x1) (- y2 y1)))

(defmacro draw-block (window x1 y1 x2 y2)
  `(xlib:draw-rectangle window *gcontext* x1 y1 (- x2 x1) (- y2 y1) t))

;;;
;; String, Window, and Font

(defmacro string (window x y string disp-font)
  `(simple-string string))
(defmacro font (font disp-font)
  `(font-width (font-height disp-font)))
(defmacro font-height (font-height disp-font))
(defmacro length (length string)
  `(length constraint* (- *x-constraint* x) max-window-width)))
(defmacro x-width (if x-width
  (font-width
    (fixed-width font
      (let ((end (if (<- (* length font-width) max-width)
        length
        (max 0 (truncate max-width font-width))))))
      (when window
        (xlib:with-gcontext (*gcontext* :font font)
          (xlib:draw-image-glyphs window *gcontext*
            x (+ y (font-ascent disp-font))
            string :end end)))
        (values (* end font-width) (+ font-height vsp)))))))
(defmacro ; this is hackish...
  (multiple-value-bind
    (end width)
    (do* ((index length (1- index))
          (width (xlib:text-width font string :end index))
          (xlib:text-width font string :end index))
         ((or (= index 0) (<= width max-width))
          (values index width)))
      (when window
        (xlib:with-gcontext (*gcontext* :font font)
          (xlib:draw-image-glyphs window *gcontext*
            x (+ y (font-ascent disp-font))
            string :end end)))
        (values width (+ font-height vsp)))))))

```

```

;;;;
;;;; Display-Item

(defstruct (display-item
  (:print-function print-display-item))
  (display-item-x item) (display-item-y item)
  (display-item-width item) (display-item-height item))

  ;;; Display-items are objects with methods to display themselves, track the
  ;;; mouse inside their boundaries, handle mouse clicks on themselves, and so
  ;;; on. Everything we put up on the screen is backed in some way by a
  ;;; Display-Item. These are the components of the total display of an object
  ;;; as described in a display-info structure.

  ;;; Redisplay-Item redraws an item (if, say, it's changed, or if its window
  ;;; has received an exposure event). If the item is the *Current-Item*,
  ;;; we call its tracker method to make sure it gets highlighted if it's
  ;;; supposed to be.

  (width height)
  (window item)
  (funcall (display-item-display item) item)

  (setf (display-item-x item) (display-item-y item))
  (setf (display-item-width item) width)
  (setf (display-item-height item) height)
  (xlib:display-force-output *display*)
  (when (and *current-item*
    (eq (display-item-window *current-item*)
        (display-item-window item)))
    (track-mouse *current-item* *mouse-x* *mouse-y*)))

  ;;; Size-Item uses the Display method to calculate the size of an item
  ;;; once displayed. If the window supplied to Display-Item is Nil, all
  ;;; the size calculation will get done, but no graphical output will
  ;;; happen.

  (defun size-item (item)
  (if (display-item-width item)
    (values (display-item-width item) (display-item-height item))
    (display-item nil 0 0)))

  ;;; Walk-Item calls the Walker method of the given Item. Walk-Item-list
  ;;; is used by some methods to walk down a list of items they have inside
  ;;; themselves.

  (defun walk-item (item function)
  (funcall (display-item-walker item) item function))

  (defun walk-item-list (list function)
  (dolist (item list)
    (when (display-item-p item)
      (walk-item item function)))))

  ;;; The Nothing-Walker is used by guys that don't have any object items
  ;;; inside them.

  (defun nothing-walker (self function)
  (declare (ignore self function)))

  ;;; Redisplay-Item invokes the Display method of an item to put it up on the
  ;;; specified window. The window, position, and size are all set, and the
  ;;; size is returned.

  (defun redisplay-item (item window x y)
  (setf (display-item-window item) window)
  (setf (display-item-x item) x)
  (setf (display-item-y item) y)
  (multiple-value-bind
    (width height)
    (funcall (display-item-display item) item window x y)
    (setf (display-item-width item) width)
    (setf (display-item-height item) height)
    (values width height)))

  ;;; Redisplay-Item redraws an item (if, say, it's changed, or if its window
  ;;; has received an exposure event). If the item is the *Current-Item*,
  ;;; we call its tracker method to make sure it gets highlighted if it's
  ;;; supposed to be.

  (defun redisplay-item (item)
  (when (display-item-window item)
    (xlib:clear-area (display-item-window item)
      :x (display-item-x item) :y (display-item-y item)
      :width (display-item-width item)
      :height (display-item-height item)))

  (multiple-value-bind

```

```

;;; Tracking and untracking.

;;; Track-Item and Untrack-Item call the right methods of the given Item.

(defun track-item (item x y)
  (funcall (display-item-tracker item) item x y))

(defun untrack-item (item)
  (funcall (display-item-untracker item) item))

;;; Update-Current-Item is used by trackers to figure out if an item
;;; is really under the mouse. If it is, and it's not the same as the
;;; *Current-Item*, the *Current-Item* gets untracked. If the mouse is
;;; inside the current item, Update-Current-Item returns T.

(defun update-current-item (item x y)
  (let ((old-current-item))
    (if (and (<- (display-item-x item) x
                (+ (display-item-x item) (display-item-width item)))
             (<- (display-item-y item) y
                  (+ (display-item-y item) (display-item-height item))))
        (setq *current-item* item)
        (setq *current-item* nil)
        (when (and old-current (not (eq *current-item* old-current)))
          (untrack-item old-current)))
    (eq item *current-item*)))

;;; The Nothing-Tracker and Nothing-Untracker don't do much.

(defun nothing-tracker (item x y)
  (update-current-item item x y))

(defun nothing-untracker (item)
  (declare (ignore item)))

;;; The Boxifying-Tracker and Boxifying-Untracker highlight and unhighlight
;;; an item by drawing or erasing a box around the object.

(defun boxifying-tracker (item x y)
  (when (update-current-item item x y)
    (boxify-item item boole-1)))

(defun boxifying-untracker (item)
  (boxify-item item boole-cl))

(defun boxify-item (item function)
  (let ((x1 (display-item-x item))
        (y1 (display-item-y item))
        (width (display-item-width item))
        (height (- (display-item-height item) 2)))
    (window (display-item-window item))
    (xlib:with-gcontext (*gcontext :function function)
      (xlib:draw-rectangle window *gcontext* x1 y1 width height)
      (xlib:display-force-output *display*)))

;;; Track-In-List tries to track inside of each item in the List.

(defun track-in-list (list x y)
  (dolist (item list)
    (when (display-item-p item)
      (when (and (<- (display-item-x item) x

```

```

;;;; Specialized Display-Item definitions.

;;;; Inspection-Items are used as the "top-level" items in the display of an
;;;; object. They've got a list of header items and a list of entry items.
;;;; (defstruct (inspection-item
;    (:print-function print-display-item)
;    (:include display-item)
;    (display 'display-inspection-item)
;    (tracker 'track-inspection-item)
;    (walker 'walk-inspection-item)
;    (:constructor make-inspection-item (objects headers entries)))
;
;    scrollbar
;    item
;  )
;
;    ;;; String-Items just have a string of text and a font that it gets displayed in.
;
;    (defstruct (string-item
;      (:print-function print-display-item)
;      (:include display-item)
;      (display 'display-string-item)
;      (:constructor make-string-item (string optional (font *entry-font))))
;
;      string
;      font
;    )
;
;    ;;; Slot-Items have a string name for the slot (e.g., structure slot name or vector
;    ;;; objects that have lots of components and so have to scroll. In addition to
;    ;;; headers and entries, they've got a scrollbar item and stuff so that the
;    ;;; entries can lazily compute where they are and what they should display.
;
;    (defstruct (scrolling-inspection-item
;      (:print-function print-display-item)
;      (:include inspection-item)
;      (:constructor make-scrolling-inspection-item
;        (tracker 'track-scrolling-inspection-item)
;        (objects headers entries scrollbar)))
;
;      scrollbar
;      display-item
;      ; To set next state
;      ; To get & increment next state
;      next
;    )
;
;    ;;; A Scrollbar-Item has buttons and a thumb bar and the stuff it needs to figure
;    ;;; out whatever it needs to figure out.
;
;    (defstruct (scrollbar-item
;      (:print-function print-display-item)
;      (:include display-item)
;      (display 'display-scrollbar-item)
;      (tracker 'track-scrollbar-item)
;      (untracker 'untrack-scrollbar-item)
;      (:constructor make-scrollbar-item
;        (first-index num-elements num-elements-displayed)
;        (next-element reset-index))
;        next
;        ; Item for which this guy's a scrollbar
;        scrollie
;        ; Y coordinate of end (back, hack)
;        bottom
;        active-button
;        ; Index of first thing to be displayed
;        first-index
;        ; Function to extract next element to be displayed
;        next-element
;        ; Function to reset internal index for next-element
;        reset-index
;        ; Max X for scrollees
;        window-width
;        ; Height of bar in pixels
;        bar-height
;        bar-top
;        bar-bottom
;        num-elements-displayed ; Number of elements displayed at once
;        num-elements ; Number of elements in scrollee
;      )
;
;    ;;; Scrolling-Items are used as the entries in Scrolling-Inspection-Items.
;    ;;; They know where
;    ;;; the object came from and how to get it again (for decaching) and how to
;    ;;; change it (for modification).
;
;    (defstruct (object-item
;      (:print-function print-display-item)
;      (:include display-item)
;      (display 'display-object-item)
;      (:constructor make-object-item
;        (tracker 'track-object-item)
;        (untracker 'untrack-object-item)
;        (mouse-handler 'mouse-object-item)
;      )
;
;      ;;; they know the scrollbar that moves them around so they can lazily do
;      ;;; their stuff.
;
```

```

(walker 'walk-object-item)
(:constructor make-object-item (object place index ref set))

object          ; The Lisp object itself
string          ; String representation cache
place           ; Place where it came from
index           ; Index into where it came from
ref             ; Function to get object, given place and index
set             ; Function to set object, given place, index and new value
)

;;; Object*-Items are like Object-items except that sometimes they can be like
;;; string items and be not-selectable.

(defstruct (object*-item
            (:print-function print-display-item)
            (:include object-item)
            (display 'display-object*-item)
            (tracker 'track-object*-item)
            (untracker 'untrack-object*-item)
            (mouse-handler 'mouse-object*-item)
            (:constructor make-object*-item (string* object live place index ref set)))
  live
  string*)

(defun display-inspection-item (self window x0 y0)
  (let ((y (+ y0 border))
        (x (+ x0 border))
        (max-width 0)
        (max-x 0)
        (first-entry-y nil)
        (header-end-y nil)
        (sb (if (scrolling-inspection-item-p self)
                (scrolling-inspection-item-scrollbar self)))))

    (when sb
      (funcall (scrollbar-item-reset-index sb) sb))
    ; First, header items.
    (when (inspection-item-Headers self)
      (dolist (item (inspection-item-Headers self))
        (multiple-value-bind (width height)
            (display-item item window x y)
            (incf y height)
            (setq max-width (max max-width width)))
        (setq header-end-y y)
        (incf y ysp)))
    (when sb
      (incf x (+ 16 border))
      (funcall (scrollbar-item-reset-index sb) sb))
    ; Then do entry items.
    (let ((max-name-width 0))
      (setq first-entry-y)
      ; Figure out width of widest entry slot name.
      (dolist (item (inspection-item-entries self))
        (when (slot-item-p item)
          (setq max-name-width (length (slot-item-name item)))))

      (dolist (item (inspection-item-entries self))
        (when (slot-item-p item)
          (unless (slot-item-max-name-width item)
            (setf (slot-item-max-name-width item) max-name-width)))
        (multiple-value-bind (width height)
            (display-item item window x y)
            (incf y height)
            (setq max-width (max max-width (+ width (length (slot-item-name item)))))))

      (setq max-x (+ x0 border max-width border))
    ; Display scrollbar, if any.
    (when sb
      (setf (scrollbar-item-bottom sb) y)
      (display-item sb window (+ x0 border) first-entry-y)
      (unless (scrollbar-item-window-width sb)
        (setf (scrollbar-item-window-width sb) (- max-width 16 border))))
    ; Finally, draw a box around the whole thing.
    (when window
      (draw-box window x0 y0 max-x y)
      (when header-end-y
        (xlib:draw-line window *gcontext* x0 header-end-y max-x header-end-y))
    ; And return size.
    (values (- max-x x0) (- (+ y border) y0)))))

(defun track-inspection-item (self x y)
  (dolist (item (inspection-item-Headers self))
    (when (and (<= (display-item-x item) x
                  (+ (display-item-y item) (display-item-width item)))
               (<= (display-item-y item) y
                  (+ (display-item-x item) (display-item-height item))))
      (track-item item x y)))

```

```

;; scrollbar item methods.

;; Yeah, we use a hard-wired constant 16 here, which is the width and height
;; of the buttons. Grody, yeah, but hey, "16" is only two keystrokes...
;; of the buttons.

(defun track-inspection-item (self x y)
  (return-from track-inspection-item nil))

(defun track-scrolling-inspection-item (self x y)
  (dolist (item (inspection-item-headers self))
    (when (and (<= (display-item-x item) x
                    (+ (display-item-x item) (display-item-width item)))
                (<= (display-item-y item) y
                    (+ (display-item-y item) (display-item-height item))))
      (track-item item x y)
      (return-from track-scrolling-inspection-item nil)))
  (let ((sb (scrolling-inspection-item-scrollbar self)))
    (if (and (<= (display-item-x sb) x (+ (display-item-x sb)
                                              (display-item-width sb)))
              (<= (display-item-y sb) y (+ (display-item-y sb)
                                              (display-item-height sb))))
        (track-item sb x y)
        (track-in-list (inspection-item-entries self) x y)))))

(defun walk-inspection-item (self function)
  (let ((*x-constraint* (if (display-item-width self)
                            (+ (display-item-x self)
                               (display-item-width self)
                               (- border))
                            (- border)))
        (max-window-width)))
    (walk-item-list (inspection-item-headers self) function)
    (walk-item-list (inspection-item-entries self) function)))

(defun draw-block window x
  (draw-bitmap window x y
    (when window
      (draw-bitmap window x y
        (if (eq (scrollbar-item-active-button self) :top)
            (if (eq (scrollbar-item-active-button self) :bottom)
                (draw-bitmap window x (- (scrollbar-item-bottom self) 16)
                  (if (eq (scrollbar-item-active-button self) :bottom)
                      (draw-bitmap window x (- (scrollbar-item-bottom self) 16)
                        (if (eq (scrollbar-item-active-button self) :bottom)
                            (*down-arrow-1* *down-arrow*)
                            (*up-arrow-1* *up-arrow*)))
                      (draw-bitmap window x (- (scrollbar-item-bottom self) 16)
                        (if (eq (scrollbar-item-active-button self) :bottom)
                            (*down-arrow-1* *down-arrow*)
                            (*up-arrow-1* *up-arrow*)))
                    (draw-bitmap window x (+ y 16) (+ x 15) (- (scrollbar-item-bottom self) 17)
                      (draw-box window x (+ y 16) (+ x 15) (- (scrollbar-item-bottom self) 17)
                        (setf (scrollbar-item-bar-top self) (+ y 17)
                          (scrollbar-item-bottom self) (- (scrollbar-item-bar-bottom self)
                            (scrollbar-item-bar-height self) (- (scrollbar-item-bar-top self)
                              (scrollbar-item-bar-bottom self)))))))
                  (draw-bitmap window x (+ y 16) (+ x 16) (- (scrollbar-item-bar-bottom self)
                    (truncate (* (- (scrollbar-item-first-index self)
                      (scrollbar-item-num-elements self))) (scrollbar-item-num-elements self)))
                    (if (eq (scrollbar-item-bar-bottom self)
                        (- (scrollbar-item-bar-bottom self)
                          (truncate (* (- (scrollbar-item-first-index self)
                            (scrollbar-item-num-elements self)))
                            (scrollbar-item-num-elements self)))
                          (scrollbar-item-bar-height self)
                          (scrollbar-item-num-elements self)))))))
                  (xlib:display-force-output *display*)))
                (values 16 (- (scrollbar-item-bottom self) y))))))

(defun track-scrollbar-item (self x y)
  (update-current-item self x y)
  (cond ((<= (display-item-self) y (+ (display-item-y self) 16))
         (setf (scrollbar-item-active-button self) :top)
         (draw-bitmap (display-item-window self) (display-item-x self) *up-arrow-1*)
         ((<= (- (scrollbar-item-bottom self) 16) y (scrollbar-item-bottom self))
          (setf (scrollbar-item-active-button self) :bottom)
          (draw-bitmap (display-item-window self) (display-item-x self) (- (scrollbar-item-bottom self) 16)
            *down-arrow-1*)))
        (t
         (untrack-scrollbar-item self))
         (xlib:display-force-output *display*)))

(defun untrack-scrollbar-item (self)
  (cond ( (eq (scrollbar-item-active-button self) :top)
         (draw-bitmap (display-item-window self) (display-item-x self) *up-arrow*))
        ((eq (scrollbar-item-active-button self) :bottom)
         (setf (scrollbar-item-active-button self) (- (scrollbar-item-bottom self) 16)
           (draw-bitmap (display-item-window self) (display-item-x self) (- (scrollbar-item-bottom self) 16)
             *down-arrow*)))
        (xlib:display-force-output *display*))
      (setf (scrollbar-item-active-button self) nil)))

```

```

;;; String item methods.

(defun display-string-item (self window x y) (string-item-font self))
  (disp-string window x y (string-item-string self))

;;; Slot item methods.

(defun display-slot-item (self window x y)
  (let ((name (slot-item-name self)))
    (name-pixel-width (* (+ 2 (slot-item-max-name-width self))
                          (font-width *entry-font*)))
    (disp-string window x y name *entry-font*)
    (multiple-value-bind (width height)
        (display-item (slot-item-object self)
                      window (+ x name-pixel-width) y)
      (values (+ name-pixel-width width border)
              (max (+ (font-height *entry-font*) vsp) height)))))

(defun track-slot-item (slot-item-object self x y)
  (track-item (slot-item-object self) x y))

(defun walk-slot-item (self function)
  (walk-item (slot-item-object self) function)
  (setf (display-item-width self)
        (+ (* (+ 2 (slot-item-max-name-width self)) (font-width *entry-font*))
           (display-item-width (slot-item-object self))
           border)))

```

;;; List item methods.

```
    ;;; If a thing in the item list is a string, we just Disp-String it.  
    ;;; That way, we don't have to cons lots of full string items all the time.  
  
(defun display-list-item (self window x0 y0)  
  (let ((x0 (sb (scrolling-item-scrollbar self)))  
        (item (scrolling-item-item self)))  
    (funcall (scrollbar-item-next-element sb) item))  
  (let ((*x-constraint* (if (scrollbar-item-window-width sb)  
                           (+ (scrollbar-item-window-width sb) x)  
                           max-window-width)))  
    (multiple-value-bind (width height)  
        (display-item item window x y)  
      (values (or (scrollbar-item-window-width sb) width)  
             height)))  
  (values (- x0) max-height))  
  
(defun track-scrolling-item (self x y)  
  (track-item (scrolling-item-item self) x y))  
  
(defun walk-scrolling-item (self function)  
  (walk-item (scrolling-item-item self) function))  
  
(defun display-item (self window x0 y0)  
  (let ((x0 (sb (scrolling-item-scrollbar self)))  
        (item (scrolling-item-item self)))  
    (funcall (scrollbar-item-next-element sb) item))  
  (let ((*x-constraint* (if (scrollbar-item-window-width sb)  
                           (+ (scrollbar-item-window-width sb) x)  
                           max-window-width)))  
    (multiple-value-bind (width height)  
        (display-item item window x y0)  
      (values (or (scrollbar-item-window-width sb) width)  
             height)))  
  (values (- x0) max-height))  
  
(defun track-list-item (self x y)  
  (track-in-list (list-item-list self) x y))  
  
(defun walk-list-item (self function)  
  (walk-item-list (list-item-list self) function))
```

```
;;; Computing display items for Lisp objects.  
  
defun compute-display-items (object * item-methods...)
```

;; Computing display items for Lisp objects.

;; For lists, what we stash in the Inspection-item-objects slot is the list of
 ;; the top level conses, rather than the conses themselves. This lets us detect
 ;; when conses "in the middle" of the list change.

```

(define plan-display-list (object)
  (cond #|((and (symbolP (car object))
    (get (car object) 'lisp::specially-grind)))
    (error "Bluer")|#
    ((or (and (< (length (prin1-to-string object)) inspect-line-length)
      (<= (length object) inspect-length))
      (= (length object) 1))
    (do ((list object (cdr list))
      (items (list "")))
      (push (make-object-item (car list) list nil 'lref 'lset) items)
      (when (not (null (cdr list)))
        (push " " items)
        (push (make-object-item (cdr list) list nil 'lref* 'lset*) items)
        (push " " items)
        (push (make-object-item (car list) list nil 'lref 'lset) items)
        (make-inspection-item
          (copy-conses object)
          nil)
        (list (make-list-item (reverse items))))
        (push " " items))
      (push " " items)
      (push (make-object-item (cdr list) list nil 'lref 'lset) items)
      (push (make-object-item (car list) list nil 'lref* 'lset*)
        (let (items nil)
          (push (make-list-item (list " "
            (make-object-item
              (car object) object nil 'lref 'lset)))
            items)
          (do ((list (cdr object) (cdr list))
            (not (consp (cdr list)))
            (cond (null (cdr list)
              (push (make-list-item (list " "
                (make-object-item
                  (car list) list nil 'lref 'lset)
                " ")))
              items)
              (t
                (push (make-list-item (list " "
                  (make-object-item
                    (car list) list nil 'lref 'lset)))
                  items)
                  (push " " items)
                  (push (make-list-item (list " "
                    (make-object-item
                      (car list) list nil 'lref 'lset)
                    " ")))
                    items))
                    (push (make-list-item (list " "
                      (make-object-item
                        (car list) list nil 'lref 'lset)))
                      items))
                      (make-inspection-item
                        (copy-conses object)
                        nil)
                        (list (make-object-item (reverse items)))))))
```

```

inspect-length
  #'(lambda (item)
    (setf (list-item-list item)
      `((, (cond ((eq cons object)
        ("")
        ((not (consp cons))
          (" " ""))
        (" " ""))
        ((if (consp cons)
          (make-object-item (car cons) cons nil 'lref 'lset)
          (make-object-item cons last 'lref* 'lset*))
        ,@ (if (or (and (eq cons last) (null (cdr cons)))
          (atom cons)
          ('" "')))
        (lincf index)
        (unless (atom cons)
          (unless (atom cons)
            (unless (atom cons (cdr cons)))
              (setq cons (cdr cons)))))))
        (#'(lambda (self)
          (setq index (scrollbar-item-first-index self))
          (setq cons (nthcdr index object)))))))
        (setf (scrollbar-item-scrolllee scrollbar)
          (make-scrolling-inspection-item
            (copy-conses object)
            nil)
          (let ((items nil))
            (dotimes (i (inspect-length)
              (setq cons (nthcdr index object)))
              (push (make-scrolling-item scrollbar (make-list-item nil))
                items)))
            (reverse items)))
            scrollbar)))
            )))
            ;;; This is kind of like (maplist #'identity list), except that it doesn't
            ;;; choke on non-Nil terminated lists.
            ;;; This is kind of like (maplist #'identity list), except that it doesn't
            ;;; choke on non-Nil terminated lists.
            (defun copy-conses (list)
              (do ((list list (cdr list))
                (conses nil))
                (atom list)
                (reverse conses)
                (push list conses)))
                (return nil)))
            (walk-item plan #'replan-object-item)
            (plan)
            (plan-dispatch (car object)))
            (t
              (defun lref (object ignore) (declare (ignore ignore)))
              (car object)
              (defun lref* (object ignore) (declare (ignore ignore)))
              (cdr object)
              (defun last (object ignore new) (declare (ignore ignore)))
              (defun first (object ignore new)
                (setf (car object) new)
                (defun last* (object ignore new) (declare (ignore ignore)))
                (setf (cdr object) new)
                (+ (length object) (if (cdr last) 1 0))))
```

```

(defun plan-display-vector (object)
  (let* ((type (type-of object))
         (length (array-dimension object 0))
         (header
           `((make-string-item (format nil ".A" (if (listp type) (car type) type)))
             ,(make-string-item (format nil ".D" length - "D" length))
             ,(make-string-item (format nil "Length = ~D" length)
                     *header-font*)
             @,(if (array-has-fill-pointer-p object)
                   *header-font*
                   ,(make-list-item (list "Fill-Pointer:
                                         (make-object-item
                                           (fill-pointer object)
                                           (make-object-item
                                             (fill-pointer object)
                                             object nil 'fpref 'fpptr)))))))
         (cond ((<= length inspect-length)
                (make-inspect-item
                  object
                  header
                  (let ((items nil)
                        (dotimes (i length)
                          (push (make-slot-item (prin1-to-string i)
                                    (make-object-item
                                      (aref object i) object i 'vref 'vset)))
                            items)))
                (reverse items))))
              (t
                (let ((scrollbar
                      (let ((index 0)
                            (make-scrollbar-item
                              0
                              length
                              inspect-length
                              #'(lambda (item)
                                  (setf (slot-item-name item) (prin1-to-string index))
                                  (let ((obj) (slot-item-object item))
                                    (setf (object-item-object obj) (aref object index))
                                    (setf (object-item-index obj) index)
                                    (setf (object-item-string obj) nil)
                                    (incf index)
                                    #'(lambda (self)
                                        (setq index (scrollbar-item-first-index self))))))
                            (setq index (scrollbar-item-scrollbar scrollbar)
                                  (make-scrolling-inspection-item
                                    object
                                    header
                                    (let ((items nil)
                                          (name-width (length (prin1-to-string (1- length))))))
                                      (dotimes (i inspect-length)
                                        (let ((slot
                                              (make-slot-item
                                                (make-object-item nil object nil 'vref 'vset)))
                                              nil
                                              (make-object-item max-name-width slot) name-width)
                                          (setf (slot-item-max-name-width slot) name-width)
                                          (push (make-scrolling-item scrollbar slot) items)))
                                        (reverse items)
                                        scrollbar)))))))
                (plan-display-object)))))

(defun replan-display-vector (plan object)
  (cond ((= (length object) (length (inspect-item-objects plan)))
         (walk-item plan #'replan-object-item
           plan-
           (c
             (plan-display object)))))


```

```

"[]"
(let ((list nil))
  (dolist (dim rev-dimensions)
    (multiple-value-bind (q r)
        (floor index dim)
      (setq index q)
      (push r list)))
  (format nil "[-D{, -D-}]" (car list) (cdr list)))))

(defun replan-display-array (plan object)
  (cond ((and (equal (array-dimensions object)
                     (array-dimensions (inspection-item-objects plan)))
              (lisp::with-array-data (data1 object)
                (lisp::with-array-data (data2 (inspection-item-objects plan))
                  (lisp::with-array-data (data3 (inspection-item-objects plan))
                    (and (eq data1 data2)
                         (= start1 start2)
                         (= end1 end2)))
                  (walk-item plan #'replan-object-item)
                  plan)
                  t)
                  (plan-display object))))))

(let ((header (make-string-item
               (format nil "Array of ~A" (array-element-type object))
               *header-font*)))
  (make-string-item
   (format nil "Dimensions = ~S" dimensions)
   *header-font*)))
  (cond ((<= length inspection-item
        (make-inspection-item
         object
         header
         (let ((items nil))
           (dotimes (i length)
             (push (make-slot-item (index-string i rev-dimensions)
                                   (make-object-item
                                    (aref data (+ start i))
                                    object (+ start i) 'vref 'vset))
                   items)
             (reverse items))))
         t)
        (let ((scrollbar
              (make-scrollbar-item
               0
               length
               inspect-length
               #'(lambda (item)
                   (setf (slot-item-name item)
                         (index-string index rev-dimensions))
                   (let ((obj) (slot-item-object item))
                     (setf (object-item-object obj)
                           (aref data (+ start index)))
                     (setf (object-item-index obj) (+ start index))
                     (setf (object-item-string obj) nil))
                   (incf index)
                   #'(lambda (self)
                       (setq index (scrollbar-item-first-index self)))))))
          (setf (scrollbar-items scrollbar)
                (make-scrolling-inspection-item
                 object
                 header
                 (let ((items nil)
                       (name-width (length (index-string (1- length)
                                                       rev-dimensions)))
                       (ddotimes (i inspect-length)
                         (let ((slot
                               (make-slot-item
                                nil
                                (make-object-item nil data nil 'vref 'vset)))
                           (setf (slot-item-max-name-width slot) name-width)
                           (push (make-scrolling-item scrollbar slot) items)))
                           (reverse items)
                           (scrollbar)))))))
        (defun index-string (index rev-dimensions)
          (if (null rev-dimensions)

```

```

(defun plan-display-structure (object)
  (let* ((dd (info type defined-structure-info (structure-ref object 0)))
         (dd-slots (dd slots))
         (make-inspection-item
          (make-inspection-item
           object
           (list (make-string-item (format nil "~A ~A"
                                             (symbol-name (c:dd-name dd)))))))
         (header-font *))
    (let ((items nil)
          (dolist (dsd dsds)
            (push (make-slot-item (c:dd-name dsd)
                                  (make-object-item
                                   (structure-ref object (c:dsd-index dsd))
                                   object (c:dsd-index dsd) 'vref 'vset)))
                  items)))
      (reverse items)))))

(defun replan-display-atomic (plan object)
  (declare (ignore object))
  (walk-item plan #'replan-object-item)
  plan)

(defun replan-object-item (item)
  (make-inspection-item
   nil
   (list (make-object-item (list object) nil 'lref 'lset)))))


```

;;; This is all very gross and silly now, just so we can get something working
 ;;; quickly. Eventually do this with a special stream that listifies things as
 ;;; it goes along...

```

(defun plan-display-text (object header stream)
  (let ((list nil))
    (do ((line (read-line stream nil nil) (read-line stream nil nil)))
        ((null line))
      (push line list)
      (setq list (reverse list))
      (if (<= (length list) inspect-length)
          (make-inspection-item
           object
           header
           (mapcar #'make-string-item list))
          (let ((index 0)
                (vector (coerce list 'vector)))
            (vector-setf-vector! vector index)
            (let ((scrollbar (make-scrollbar-item
                              0 (length list) inspect-length
                              #'(lambda (item)
                                  (setf (string-item-string item)
                                        (aref vector index))
                                  (incf index)
                                  #'(lambda (self)
                                      (setq index
                                            (scrollbar-item-first-index self))))))
              (setf (scrollbar-item-scrollee scrollbar)
                    (make-rolling-inspection-item
                     object
                     header
                     (let ((items nil)
                           (dotimes (i inspect-length)
                             (push (make-scrolling-item scrollbar
                                           (make-string-item "")))
                           (nreverse items)))
                     (reverse items)))
                     scrollbar)))))))

```

```

;; Displaying old and new plans in old and new windows.

(defun new-plan-in-new-display (object plan &optional name)
  (multiple-value-bind (width height) (size-item plan)
    (if add-border
        (incf width 10)
        (incf height 10))
    (multiple-value-bind (x y) (next-window-position width height)
      (let* ((window (xlib:create-window :parent *root* :x x :y y
                                         :width width :height height
                                         :background "white-pixel"
                                         :border-width 2)))
        (display-info (make-display-object plan window))
        (xlib:set-m-properties window
          :name "Inspector Window"
          :icon-name "Inspector Display"
          :resource-name "Inspector"
          :x x :y y :width width :height height
          :user-specified-size-p t
          :user-specified-position-p t
          :min-width width :min-height height
          :width-inc nil :height-inc nil)
        (add-window-display-info-mapping window display-info)
        (xlib:map-window window)
        (xlib:clear-area window)
        (xlib:with-state (window) important-xevents-mask)
        (setf (xlib:window-event-mask window) *cursor*)
        (setf (xlib:window-cursor window) *cursor*)
        (xlib:display-finish-output *display*)
        (display-item plan window 5)
        (push display-info *display-infos*)
        (multiple-value-bind (x y same-screen-p child mask root-x root-y root)
            (declare (ignore same-screen-p child mask root-x root-y root))
            (when (and (< 0 x (+ width 10)) (< 0 y (+ height 10)))
                  (track-mouse plan x y))
            (xlib:display-force-output *display*)
            (display-info)))
      (display-create-display-of-object (object &optional name)
        (new-plan-in-new-display object (plan-display-object) name)
        (unless (eq new-old)
          (setf (display-item-info-display-item display-info) new)
          (let ((window (display-info-window display-info)))
            (when (and *current-item*
                      (eql (display-item-window *current-item*) window))
              (setq *current-item* nil)
              (multiple-value-bind (width height)
                  (size-item new)
                (xlib:with-state (window)
                  (setf (xlib:drawable-width window) (+ width 10))
                  (setf (xlib:drawable-height window) (+ height 10))
                  (xlib:clear-area window
                    (display-item-new window 5)
                    (setf (display-item-window new) window
                      (display-item-x new) 5
                      (display-item-y new) 5
                      (display-item-width new) width
                      (display-item-height new) height)
                    (xlib:display-force-output *display*)))))
        (cond ((eq object (display-info-object display-info))
               (new-plan-in-old-display display-info
                 (display-item-info-display-item display-info)
                 (repaint-display
                   (display-item display-info))))))

(defun update-display-of-object (display-info
  &optional
  (object (display-info-object display-info)))
  (cond ((eq object (display-info-object display-info)
            (new-plan-in-old-display display-info
              (display-item-info display-item display-info))))))

(t
  (setf (display-info-object display-info) object)
  (new-plan-in-old-display display-info
    (display-item-info display-item display-info)
    (repaint-display
      (display-item display-object)))))

(xlib:display-force-output *display*)

;;; DELETING-WINDOW-DROP-EVENT checks for any events on win. If there is one,
;;; it is removed from the queue, and t is returned. Otherwise, returns nil.
;;; (defun deleting-window-drop-event (display win)
;;;
;;;(xlib:display-finish-output display)
;;;(let ((result nil))
;;;(xlib:process-event
;;;  display :timeout 0
;;;  :handler #'(lambda (key event-window window &allow-other-keys)
;;;    (if (or (eq event-window win) (eq window win)
;;;             (setf result t)
;;;             nil)))
;;;
;;;  result))

(defun remove-display-of-object (display-info)
  (let ((window (display-info-window display-info)))
    (setf (xlib:window-event-mask window) #.(xlib:make-event-mask))
    (xlib:display-finish-output *display*)
    (loop (unless (deleting-window-drop-event *display* window) (return)))
    (xlib:destroy-window window)
    (xlib:display-finish-output *display*)
    (delete-window-drop-event *display* window)
    (setq *display-infos* (delete display-info *display-infos*))))
```

```

;; The command interpreter.

((or (eq key-event #k"q") (eq key-event #k"U"))
;; Quit.
  ((try-to-quit)
   (or (eq key-event #k"p") (eq key-event #k"P"))
       ;; Proceed.
       ((try-to-proceed)
        (or (eq key-event #k"r") (eq key-event #k"R"))
            ;; Recompute object (decache).
            ((update-display-of-object display-info)
             (or (eq key-event #k"u") (eq key-event #k"U"))
                 ;; Up (pop history stack).
                 ((pop history-stack)
                  (when (display-info-stack display-info)
                    (let ((parent (pop display-info-stack display-info))
                          (setf (display-info-object display-info) (car parent))
                          (new-plan-in-old-display display-info)
                          (display-in-old-item display-info)
                          (cadr parent))
                      (update-display-of-object display-info)))
                    (or (eq key-event #k"Leftdown")
                        (eq key-event #k"Middledown")
                        (eq key-event #k"Rightdown")
                        (eq key-event #k"Super-Leftdown")
                        (eq key-event #k"Super-Middledown")
                        (eq key-event #k"Super-Rightdown")
                        (eq key-event #k"super-Leftdown")
                        (eq key-event #k"super-Middledown")
                        (eq key-event #k"super-Rightdown")
                        (when *current-item*
                          (funcall (display-item-house-handler *current-item*)
                                  *current-item* display-info key-event)))))))
            (when *current-item*
              (funcall (display-item-house-handler *current-item*)
                      *current-item* display-info key-event))))))

(defvar *can-quit* nil)
(defvar *can-processed* nil)
(defvar *unwinding* t)

(defun try-to-quit ()
  (setq *current-item* nil)
  (when *can-quit*
    (setq *unwinding* nil)
    (ext:flush-display-events *display*)
    (throw 'inspect-quit nil)
    (try-to-processed)))

(defun try-to-processed ()
  (when *can-processed*
    (setq *unwinding* nil)
    (ext:flush-display-events *display*)
    (throw 'inspect-processed nil)))

(defvar *do-command* nil)

(defun do-command (display-info key-event)
  (cond (*do-command*
         (funcall *do-command* display-info key-event))
        ((or (eq key-event #k"r") (eq key-event #k"D"))
         ;; Delete current window.
         (remove-display-of-object display-info)
         (setq *current-item* nil)
         (unless *display-infos*
           (try-to-quit)
           (try-to-processed)))
        ((or (eq key-event #k"h") (eq key-event #k"H") (eq key-event #k"?"))
         ;; Modify something.
         ;; Since the tracking stuff sets up event handlers that can throw past
         ;; the CLX event dispatching form in INSPECTOR-EVENT-HANDLER, those
         ;; handlers are responsible for discarding their events when throwing
         ;; to this CATCH tag.
         ;;
         (catch 'quit-modify
           (let* ((destination-item (track-for-destination))
                  (source (cond
                            (eq key-event #k"m")
                            (object-item-object (track-for-source))))))
             (t
              (format *query-io*
                     "~>Form to evaluate for new contents: ")
              (force-output *query-io*)
              (eval (read *query-io*)))
              (funcall (object-item-set destination-item)
                      (object-item-place destination-item)
                      (object-item-index destination-item)
                      source)
              (update-display-of-object display-info))))))

;; The command interpreter.

```



```
(funcall (scrollbar-item-reset-index self) self)
(dolist (item (scrolling-inspection-item-entries
  (scrollbar-item scrollbar-item scrollbar self)))
  (redisplay-item item))
  (redisplay-item self)))

(define track-mouse (item x y)
  (track-item item x y))
  (track-item item x y))

(define scrollbar-item-reset-index (index)
  (funcall (scrollbar-item-reset-index self) self)
  (scrollbar-item-reset-index scrollbar-item scrollbar self)
  (redisplay-item scrollbar-item)
  (redisplay-item self)))
```

```
;; Top-level program interface.
```

```
(defun show-object (object &optional name)
  (inspect-init)
  (dolist (display-info *display-infos*)
    (when (if name
              (eq name (display-info-name display-info))
              (eq object (display-info-object display-info)))
      (update-display-of-object display-info object)
      (return-from show-object nil)))
    (create-display-of-object object name)))

(defun remove-object-display (object &optional name)
  (dolist (display-info *display-infos*)
    (when (if name
              (eq name (display-info-name display-info))
              (eq object (display-info-object display-info)))
      (remove-display-of-object display-info)
      (return nil)))

  (define remove-all-displays ()
    (dolist (display-info *display-infos*)
      (remove-display-of-object display-info)))
```

```

;;; top-level user interface.

(defvar *interface-style* :graphics
  "This specifies the default value for the interface argument to INSPECT. The
  default value of this is :graphics, indicating when running under X, INSPECT
  should use a graphics interface instead of a command-line oriented one.")

(defun inspect (optional (object nil) object-p)
  (interface *interface-style*)
  "This function allows the user to interactively examine Lisp objects.
  Interface indicates whether this should run with a :graphics interface or a
  command-line oriented one; of course, when running without X, there is no
  choice. Supplying :window, :windows, :graphics, :graphical, and :x gets a
  windowing interface, and supplying :command-line or :tty gets the other
  style. Invoking this with no arguments resumes inspection of items left
  active from previous uses, but this only works when running under X."
  (cond ((or (member 'interface '(command-line :tty))
    (not (assoc :display ext:*environment-list*)))
    (when object-p (tty-inspect object)))
    (not (member 'interface '(:window :windows :graphics :graphical :x)))
    (error "Interface must be one of :window, :windows, :graphics,
    :graphical, :x, :command-line, or :tty -- not ~S."
      interface))
  (object-p
    (inspect-init)
    (let ((dismbodyl-display-infos nil)
      (*inspect-result* object)
      (*x-constraint* max-window-width)
      (*can-quit* t)
      (*can-proceed* t))
    (let ((*display-infos* nil)
      (create-display-of-object object)
      (catch 'inspect-proceed
        (unwind-protect
          (progn
            (catch 'inspect-exit
              (loop (system:serve-event)))
            (loop (unwinding t))
            (when *unwinding*
              (do ((display-info (pop *display-infos*)
                (pop *display-infos*))
                ((null display-info))
                (remove-display-of-object display-info)))
                (setq disembodied-display-infos *display-infos*))
              (dolist (display-info (reverse disembodied-display-infos))
                (push display-info *display-infos*))
              (*inspect-result*)
              (*display-infos*)
              (inspect-init)
              (let ((*inspect-result* nil)
                (*can-quit* t)
                (*can-proceed* t))
                (catch 'inspect-proced
                  (catch 'inspect-exit
                    (loop (system:serve-event)))
                  *inspect-result*)
                  (t (error "No object supplied for inspection and no previous ~
                    inspection object exists."))))
```

; Only works if inspect has previously been called, so that the
; fontname and stuff is set up
; Otherwise, it shows the help file passed in thru a new window

```
(defun showHelpFile (filename)
  (let ((inspect-length (max inspect-length 12)))
    (with-open-file (stream filenn :direction :input)
      (new-plan-in-new-display
        nil
        (plan-display-text nil
          (list (make-string-item "Help" *header-font*)))))
```

```
*****  
** FANSYS: A Computer Model of Text Comprehension and Question  
** Answering for Failure Analysis  
**  
** File: garnload.lsp  
** Developed by: Sergio J. Alvarado  
** Ronald K. Braun  
** Kenrick J. Mock  
**  
** Artificial Intelligence Laboratory  
** Computer Science Department  
** University of California  
** Davis, CA 95616  
**  
** Funds for the support of this study have been allocated by the  
** NASA-Ames Research Center, Moffett Field, California, under  
** Interchange No. NCA2-721.  
*****
```

```
*** Load Garnet Code  
*** This may take a while.  
*** You will have to insert the proper pathname here.  
(load "/net/epoch/usr/eceugrad/mock/garnet/garnet-loader")
```

```

Failure Cause
(*Failure Detection Procedure* M-Failure-Detection)
(*Failure Correction Procedure* M-Failure-Correction)
(*Failure Steps* M-Detection-And-Group)
(*Correction Steps* M-Correction-And-Group))
(defvar *FailedCompConv* '(
  ("Unknown" nil) ("Ring Concentrator" M-SDP)
  ("Gateway" M-G) ("Standard Data Processor" M-SDP)
  ("Time Generation Unit" M-TGU) ("Mass Storage Device" M-MSD))
; This one's a bit different. Use the Mode Global Var and the
; Name of the button to check for.
; Name of the button to check for.
(defvar *GivenConv* '(
  ("Failure Mode..." (*FailedModeConv* FailedModeButtons))
  ("Failure Cause..." (*FailedCauseConv* FailedCauseButtons))
  ("Failure Detection..." (*FailedDetectConv* FailedDetectButtons))
  ("Failure Correction..." (*FailedCorrConv* FailedCorrButtons)))
))

(defvar *FailedModConv* '(
  ("Loss of output - failure to start" M-StartUp-no-output)
  ("Loss of output - failure during operation" M-operational-no-output)
  ("Operational - Erroneous Output" M-operational-erroneous-output)))
(defvar *FailedCauseConv* '(
  ("Thermal Shock" M-Thermal-Shock-Cause)
  ("Piece-Part Failures" M-faulty-component-cause)
  ("Contamination" M-contamination-cause)
  ("Temperature (High or Low)" M-temperature-out-of-range-cause)
  ("Mechanical Shock" M-mechanical-shock-cause)
  ("Erroneous Input" M-erroneous-input)))
(defvar *FailedDetConv* '(
  ("Next Node Detection" M-Next-node-detection)
  ("System Management Heartbeat Messages" M-heartbeat-detection)
  ("Error Correction Code Detection" M-ecc-detection)
  ("Power-On Self Test" M-post-detection)))
(defvar *FailedCorConv* '(
  ("Check Connections or Replace" M-non-operational-replacement)
  ("Automatic Backup from Lookup-Table" M-lookup-backup)
  ("Network Self-Reconfiguration" M-reconfiguration)
  ("Replace with Spare" M-replace-with-spare)
  ("Switch or Bypass via Backup Net" M-switch-or-bypass)))
)

; See Garnet documentation for further details on how the Garnet
; specific functions are working.
; Load Garnet buttons
; Functions or "button handlers" have been set up for each button
; type. These functions are called when their respective buttons
; are clicked.
(garnet-load "gadgets:x-buttons-loader")
(garnet-load "gadgets:text-buttons-loader")
(in-package "USER" :use '("LISP" "KR"))
; Initialization Stuff
; Global variable used to hold list of things selected in the "Given" slot
; and also the "Requested" slot
(defvar *given* nil)
(defvar *requested* nil)

; Lists for converting button values -> mop names
; This makes it easy to modify the name of a button - just change it
; in this list definition.

(defvar *ReqConv* '(
  ("ORU" M-ORU) ("Failure Mode" M-Failure-Mode)))

```

```

Create a necessary font
::::::::::::::::::
; Create-Instance 'Qbifont opal;font
  (:family :sans-serif) (:face :roman) (:size :very-large)
; Create-Instance 'Qmefont opal;font
  (:family :sans-serif) (:face :roman) (:size :large)
; Create-Instance 'Qagf font
  (:family :sans-serif) (:face :roman) (:size :large)

; Make Windows
; Window and aggregate for getting Requested Info
  (:create-instance 'QWin interactractor-window
    (:left 120) (:top 30) (:width 539) (:height 600) (:visible nil)
    (create-instance 'QAgg opal:aggregate)
    (s-value QWin :aggregate QAgg)
    (s-value QWin :aggregate QAgg)

; Window for quitting the program
  (:create-instance 'QuitWin interactractor-window
    (:left 110) (:top 300) (:width 30) (:height 30)
    (create-instance 'QAgg opal:aggregate)
    (s-value QuitWin :aggregate QuitAgg)
    (s-value QuitWin :aggregate QuitAgg)
    (s-value QuitWin :height 30)
    (s-value QuitWin :height 15)
    (s-value QuitWin :height 15))

; Window for starting the question menu function
  (:create-instance 'StartWin interactractor-window
    (:left 1100) (:top 760) (:width 147) (:height 15)
    (create-instance 'StartAgg opal:aggregate)
    (s-value StartWin :aggregate StartAgg)
    (s-value StartWin :height 30)
    (s-value StartWin :height 15)
    (s-value StartWin :height 15))

; Window for asking about help
  (:create-instance 'HelpWin interactractor-window
    (:left 1100) (:top 868) (:width 48) (:height 15)
    (create-instance 'HelpAgg opal:aggregate)
    (s-value HelpWin :aggregate HelpAgg)
    (s-value HelpWin :height 30)
    (s-value HelpWin :height 15)
    (s-value HelpWin :height 15))

; Window for doing the memory inspector
  (:create-instance 'InspectWin interactractor-window
    (:left 1100) (:top 825) (:width 131) (:height 15) (:visible 't))
    (create-instance 'InspectAgg opal:aggregate)
    (s-value InspectWin :aggregate InspectAgg)
    (s-value InspectWin :height 30)
    (s-value InspectWin :height 15)
    (s-value InspectWin :height 15))

; Make text
; Given Window text
; Requested Text
; Create-Instance 'ReqCont opal:text
  (:left 140) (:top 15) (:string "Select the Requested Items")
  (s-value Qbigfont)
  (opal:add-components QAgg ReqText)

; Failed Component Window text
; Create-Instance 'FailedCompText opal:text
  (:left 120) (:top 15) (:string "Select the Failed Component")
  (s-value Qbigfont)
  (opal:add-components QAgg FailedCompText)

; Failure Mode Window text
; Create-Instance 'FailedModeText opal:text
  (:left 140) (:top 15) (:string "Select the Failure Mode")
  (s-value Qbigfont)
  (opal:add-components QAgg FailedModeText)

; Failure Cause Window text
; Create-Instance 'FailedCauseText opal:text
  (:left 130) (:top 15) (:string "Select the Failure Cause")
  (s-value Qbigfont)
  (opal:add-components QAgg FailedCauseText)

; Failure Detection Window text
; Create-Instance 'FailedDetectText opal:text
  (:left 120) (:top 15) (:string "Select Failure Detection")
  (s-value Qbigfont)
  (opal:add-components QAgg FailedDetectText)

; Failure Correction Window text
; Create-Instance 'FailedCorrText opal:text
  (:left 120) (:top 15) (:string "Select Failure Correction")
  (s-value Qbigfont)
  (opal:add-components QAgg FailedCorrText)

; Make Buttons
; Buttons for Requested Information
  (:create-instance 'ReqButtons gg:x-button-panel
    (:left 90) (:top 80)
    (s-selection-function 'ReqHandler)
    (items (GetButtonStrings *ReqCony*))
    (opal:add-components QAgg ReqButtons)
    (g-value ReqButtons :value) ; Need to get to initialize slot)

; Button to Continue in the Requested Window
  (:create-instance 'ReqContButton gg:text-button
    (:left 210) (:top 50)
    (s-selection-function 'ReqConHandler)
    (string "Click to Continue")
    (s-selection-function 'ReqConHandler)
    (opal:add-components QAgg ReqContButton)
    (g-value ReqContButton :value))

; Buttons for Failed Component
; Create-Instance 'FailedCompButtons gg:text-button-panel
  (:left 180) (:top 80)
  (items (GetButtonStrings *FailedCompCony*))
  (s-selection-function 'FailedCompHandler)
  (opal:add-components QAgg FailedCompButtons)
  (g-value FailedCompButtons :value)

; Button to select the Given Information
; Create-Instance 'GivenButtons gg:x-button-panel
  (:left 30) (:top 15)
  (string "Click on the information given in the question")

```

```

(:left 1) (:top 1)
  (:string "Enter Search Query")
  (:create-instance 'HelpButton gg:text-button
    (:selection-function 'StartWindow)
    (opal:add-components StartAgg StartSearchButton)
    (opal:update StartWin)

    ; Button to get Help
    (:left 1) (:top 1)
    (:create-instance 'HelpButton gg:text-button
      (:selection-function 'HelpHandler)
      (:string "Help")
      (opal:add-components HelpAgg HelpBButton)
      (opal:update HelpWin)

    )

    ; Button to start the Inspector on M-ROOT
    (:left 1) (:top 1)
    (:create-instance 'InspectButton gg:text-button
      (:selection-function 'InspectHandler)
      (:string "Memory Inspector")
      (opal:add-components InspectAgg InspectButton)
      (opal:update InspectWin)

    )

    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ; Start Function to start showing sequences of windows
    ; This is the main function which will be called
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    (defun StartWindow(&optional p1 p2)
      (setf *given* nil)
      (setf *requested* nil)
      (s-value QWin :visible 't)
      (s-value ReqOnButton :value nil)
      (s-value ReqOffButton :value nil)
      (s-value StartSearchButton :value nil)
      (opal:update StartWin)

      (s-value ReqButtons :value nil)
      (s-value FailedCompButtons :value nil)
      (s-value FailedContButton :value nil)
      (s-value GivenButtons :value nil)
      (s-value FailedModeButtons :value nil)
      (s-value FailedCauseButtons :value nil)
      (s-value FailedCauseContButton :value nil)
      (s-value FailedDetectedButtons :value nil)
      (s-value FailedCorrButtons :value nil)
      (s-value ReqText :visible 't)
      (s-value FailedCompText :visible nil)
      (s-value FailedContText :visible nil)
      (s-value FailedCauseText :visible nil)
      (s-value FailedModeText :visible nil)
      (s-value FailedCauseContText :visible nil)
      (s-value RedContButton :visible 't)
      (s-value ReqButtons :visible 't)
      (s-value FailedCompButtons :visible nil)
      (s-value GivenButtons :visible nil)
      (s-value GivenContButton :visible nil)
      (s-value FailedModeButtons :visible nil)
      (s-value FailedCauseButtons :visible nil)
      (s-value FailedCauseContButton :visible nil)
      (s-value FailedDetectedButtons :value nil)
      (s-value FailedCorrButtons :value nil)
      (s-value ReqText :visible 't)
      (s-value FailedCompText :visible nil)
      (s-value FailedContText :visible nil)
      (s-value FailedCauseText :visible nil)
      (s-value FailedModeText :visible nil)
      (s-value FailedCauseContText :visible nil)
      (s-value FailedDetectedText :visible nil)
      (s-value FailedCorrText :visible nil)
      (s-value ReqButtons :visible 't)
      (s-value FailedCompButtons :visible nil)
      (s-value GivenButtons :visible nil)
      (s-value GivenContButton :visible nil)
      (s-value FailedModeButtons :visible nil)
      (s-value FailedCauseButtons :visible nil)
      (s-value FailedCauseContButton :visible nil)
      (s-value FailedDetectedButtons :visible nil)
      (s-value FailedCorrButtons :visible nil)
      (s-value ReqText :visible 't)
      (s-value FailedCompText :visible nil)
      (s-value FailedContText :visible nil)
      (s-value FailedCauseText :visible nil)
      (s-value FailedModeText :visible nil)
      (s-value FailedCauseContText :visible nil)
      (s-value FailedDetectedText :visible nil)
      (s-value FailedCorrText :visible nil)
      (s-value ReqButtons :visible 't)
      (s-value FailedCompButtons :visible nil)
      (s-value GivenButtons :visible nil)
      (s-value GivenContButton :visible nil)
      (s-value FailedModeButtons :visible nil)
      (s-value FailedCauseButtons :visible nil)
      (s-value FailedCauseContButton :visible nil)
      (s-value FailedDetectedButtons :visible nil)
      (s-value FailedCorrButtons :visible nil)
      (opal:update QWin)

    )

    ; Buttons for Failure Mode
    (:left 1) (:top 1)
    (:create-instance 'FailedModeButtons gg:text-button
      (:selection-function 'FailedModeConv*)
      (:left 110) (:top 80)
      (:items (GetButtonStrings *FailedModeConv*)
        (opal:add-components FailedModeHandler)
        (:selection-function 'FailedModeHandler)
        (g-value FailedModeButtons :value)

      )

      ; Buttons for Failure Cause
      (:left 160) (:top 80)
      (:items (GetButtonStrings *FailedCauseConv*)
        (opal:add-components FailedCauseHandler)
        (g-value FailedCauseButtons :value)

      )

      ; Button to Continue in the Failure Cause Window
      (:left 250) (:top 500)
      (:create-instance 'FailedCauseContButton gg:text-button
        (:selection-function 'FailedCauseHandler)
        (:string "Return")
        (opal:add-components FailedCauseContButton)
        (g-value FailedCauseContButton :value)

      )

      ; Buttons for Failure Detection Window
      (:left 110) (:top 80)
      (:items (GetButtonStrings *FailedDetectConv*)
        (opal:add-components FailedDetectHandler)
        (g-value FailedDetectButtons :value)

      )

      ; Buttons for Failure Correction Window
      (:left 140) (:top 80)
      (:items (GetButtonStrings *FailedCorrConv*)
        (opal:add-components FailedCorrHandler)
        (:selection-function 'FailedCorrHandler)
        (g-value FailedCorrButtons :value)

      )

      ; Button to Quit the program entirely
      (:left 1) (:top 1)
      (:string "Quit Program")
      (:selection-function 'MyQuit)
      (opal:add-components QuitAgg QuitButton)
      (opal:update QuitWin)

    )

    ; Button to Start the Search
    (:create-instance 'StartSearchButton gg:text-button
      (:selection-function 'StartWindow)
      (opal:update StartWin)
    )
  )

```

((My-Member ***Correction Steps*** *requested*)

;;;;;; Button Handlers for the various Windows
;;;;;; Button Handlers for the various Windows

; Handle Requested click. Need to check if the person clicks on
; the correction steps or detection steps, and move on from there.
; See GivenHandler for more info

(defun ReqHandler (p1 p2)

(let ((temp nil))
(cond

((> (length *requested*) (length p2)) ; New click~"Un-X'd" a box
(setf *requested* p2)) ; Find button clicked upon

((setf temp (FindMissing *requested* p2)) ; Find button clicked upon
(setf *requested* p2))

(cond

((My-Member ***Detection Steps*** *requested*)

(cond
(> (length *requested*) 1) ;(format t "*** Warning: Detection Steps selection overriding previous ~e~")
(format t "*** Warning: Detection Steps selection must be selected~e~")
(format t " individually.~e~")

(format t " (s-value ReqButtons :value '(*Detection Steps*))

(setf *requested* '(*Detection Steps*))

(opal:update QWin))

((My-Member ***Correction Steps*** *requested*)

(cond
(> (length *requested*) 1) ;(format t "*** Warning: Correction Steps selection overriding previous ~e~")
(format t "*** Warning: Correction Steps selection. Correction steps must be selected~e~")
(format t " individually.~e~")

(format t " (s-value ReqButtons :value '(*Correction Steps*))

(setf *requested* '(*Correction Steps*))

(opal:update QWin))

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)


```
((null Givens) nil)
(t
  (setf temp (g-value (eval (caddr Givens)) :value)))
  (cond ((atom temp) (setf temp (list temp)))
        ((cons (car (GetAllConversions (eval (caar Givens))) temp))
           (cons (car (GetAllConversions (eval (cadr Givens))))))))
    (FindGiven (cdr Givens)))))

; Quit program entirely
(defun MyQuit (p1 p2)
  (quit))
```

```

***** FANSYS: A Computer Model of Text Comprehension and Question Answering for Failure Analysis *****

File: title.lsp
Developed by: Sergio J. Alvarado
                Ronald K. Braun
                Kendrick J. Mock
Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616
Funds for the support of this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.

(in-package "USER" :use ("LISP" "KR"))

(setf *title*
      (create-instance 'titlewin
                      :interactor-window
                      (create-instance 'titlemp opal:bitmap
                                      (:left 2) (:top 300) (:justification :center) (:font medfont)
                                      (:line-style opal:orange-line)
                                      (:string "Sergio Alvarado")
                                      ;Ronald Braun
                                      ;Kenrick Mock
                                      ;(click mouse to Continue")
                                      ;(opal:add-components topagg fantext)
                                      ;(opal:components topagg authortext)
                                      ;(opal:update titlewin)
                                      ;Wait for person to click mouse on title window to make it go away
                                      ;(create-instance nil inter:button-interactor
                                      ;(:window titlewin)
                                      ;(:start-where (list :in titlewin))
                                      ;(:continuous nil)
                                      ;(:final-function #'(lambda (inter final-ob) (over)
                                      ;(opal:destroy titlewin))))))

(opal:add-components topagg *title*)
(s-value titlewin :visible 't)
(opal:update titlewin)
(s-value titlewin :visible nil)
(opal:update titlewin)
(s-value titlewin :visible 't)
(opal:update titlewin)

; (create-instance 'blqfont opal:font
;   (:family :sans-serif) (:face :roman) (:size :very-large))
;   ; What follows is the old FANSYS logo code to display
;   ; the names and info of the authors. This has now been
;   ; incorporated directly into the bitmap which is
;   ; displayed above, so the following is no longer necessary.
;   ; However, I have not deleted it in the event that we would
;   ; like to display some other text on the title window.

; (create-instance 'medfont opal:font
;   (:family :sans-serif) (:face :roman) (:size :large))
;   ; (create-instance 'fanteext opal:text
;   ; (:string "FANSYS - Failure Analysis System")
;   ; (:left 80) (:top 250)
;   ; (:line-style opal:white-line)
;   ; (:font blqfont))

```

```

/*
 * FANSYS: A Computer Model of Text Comprehension and Question
 * Answering for Failure Analysis
 *
 * File: tclient.lsp
 */
* Developed by: Sergio J. Alvarado
* Ronald K. Braun
* Kenrick J. Mock
*
* Artificial Intelligence Laboratory
* Computer Science Department
* University of California
* Davis, CA 95616
*
* Funds for the support of this study have been allocated by the
* NASA-Ames Research Center, Moffett Field, California, under
* Interchange No. NCA2-721.
*
***** */

;;;
;; Code to connect to the server.
;;;

;;;
;; Connect to trace server. Be sure to use proper host name.
;; Port 4186 has been chosen as the port for the trace.
;; Need to add error handling sometime. Be sure to start the
;; server first.
;;
;; The wire package is part of the CMU Common Lisp system.
;; CMU Common Lisp is available by FTP from lisp-rt1.sllisp.cs.cmu.edu

(format t "~~Trace Client - Make sure server process is running before starting.~<")
;; Global variable to hold the port file descriptor
(defvar *tracewire* (wire:connect-to-remote-server "toadflax" 4186))

(format t "Connect successful. Ready and waiting for trace input.~<")

;; Just loop forever here getting input and printing it
;; Should add code to have it quit if nil is returned, but this will
;; require setting up an error handler, as the thing is expecting a string.
;;
;; (prog ((val nil)
;;        (looplabel
;;          (setf val (wire:wire-get-string *tracewire*))
;;          (wire:wire-output-string *tracewire* "ack")
;;          (wire:wire-force-output *tracewire*)
;;          (princ val)
;;          (go looplabel)))

```

;;; Prompt the user to open up another window. Wait until the window is

;;; opened before continuing.

;; FANSYS: A Computer Model of Text Comprehension and Question
;; Answering for Failure Analysis

;; File: tserver.lsp

;; Developed by: Sergio J. Alvarado

;; Ronald K. Braun

;; Kenrick J. Mock

;; Artificial Intelligence Laboratory

;; Computer Science Department

;; University of California

;; Davis, CA 95616

;; Funds for the support of this study have been allocated by the

;; NASA-Ames Research Center, Moffett Field, California, under

;; Interchange No. NCA2-721.

;; *****

;; Server to send output to another CMU Lisp process running a trace

;; window.

;; *****

;; Global variable to hold file descriptor for port

(defvar *tracewire* nil)

;; Set up a handler that will set the global variable of the wire

;; when someone connects to the server.

;;(defun tracehandler (wire hostaddr)

(format t "Connection established with wire ~S-~A~" wire

(ext:host-entry-name (ext:lookup-host-entry hostaddr))

(setf *tracewire* wire))

;; set up server for trace using port 4186

(defvar *traceserv* (wire:create-request-server 4186 'tracehandler))

;; Call this when done. Actually, it's not necessary, could just quit

;; ST - Send Trace

;; Lisp to free the port.

;;(defun Shut-Trace-Server ()

(wire:destroy-request-server *traceserv*))

;; Call this function to send output to the trace window/client

;; Sends data as a string, then wait for acknowledgement of dataset.

(defun st (obj)

(cond

((stringp obj) (wire:wire-output-string *tracewire*

(format nil "~A-~A~" obj))) (format nil "~S-~S~" obj)))

(t (wire:wire-output-string *tracewire* (format nil "~S-~S~" obj))))

(t (wire:wire-force-output *tracewire*) ; flush buffer

(wire:wire-get-string *tracewire*) ; wait for ack

)

```

*****  

/* FANSYS: A Computer Model of Text Comprehension and Question  

 * Answering for Failure Analysis  

 *  

/* File: qa.shell  

/* Developed by: Sergio J. Alvarado  

/* Ronald K. Braun  

/* Kenrick J. Mock  

/*  

/* Artificial Intelligence Laboratory  

/* Computer Science Department  

/* University of California  

/* Davis, CA 95616  

/*  

/* Funds for the support of this study have been allocated by the  

/* NASA-Ames Research Center, Moffett Field, California, under  

/* Interchange No. NCA2-721.  

/*  

*****  

/*  

/* FANSYS Project  

/* File: qa.shell  

/* Author: Ron Braun  

/*  

/* This file implements an extremely simple question shell. The user may  

/* type in a question in English; this shell interfaces the question  

/* to the parser.  

/*  

(defun start-shell ()  

  (setf *mode* 'qa)  

  (format t "~>Starting the question shell...")  

  (format t "~>")  

  (format t ">")  

  (do ((question (read-delimited-list #\?)) (read-delimited-list #\?)))  

      ((nil))  

    (setf question (append question (list ',*q-mark*)))  

    (format t "~-s" question)  

    (parse question)  

    (format t "~-s")  

  )  


```



```

(failed-component m-RC FAILED-NODE)
  (cause m-thermal-shock-cause instance)
  (failed-component m-RC FAILED-NODE)
    (detection m-next-node-detection instance)
      (failed-component m-RC FAILED-NODE)
        (detection-component m-rc DETECTION-NODE)
          (detect m-token instance)
            (message m-switch-or-bypass instance)
              (correction m-RC FAILED-NODE)
                (failed-component m-RC FAILED-NODE)
                  (correction-component m-sm instance)
                    (configuration m-core-network instance NETWORK)
                      (configuration m-core-network instance NETWORK)
                        (backup-configuration m-backup-rc instance)
                          (backup-component m-backup-rc instance)
                            (system m-DMS instance)
                              (framework m-short-term instance)

CORRECTION-PLAN
  (correction m-replace-with-spare instance)
    (failed-component m-RC FAILED-NODE)
      (correction-component m-crew instance)
        (configuration m-core-network NETWORK)
          (configuration m-core-network NETWORK)
            (backup-component m-backup-rc instance)
              (framework m-long-term instance)
                (effect m-failure-effect instance)
                  (sequence-of-steps m-procedure CORRECTION-PLAN)
                    (effected-component m-crew instance)
                      (effect m-failure-effect instance)
                        (sequence-of-steps i-m-null-procedure)
                          (effected-component m-mission-support instance)
                            (effect m-failure-effect instance)
                              (sequence-of-steps i-m-null-procedure)
                                (effected-component m-systems instance)
                                  (effect m-failure-effect instance)
                                    (sequence-of-steps i-m-null-procedure)
                                      (sequence-of-steps i-m-null-procedure)
                                        (effected-component m-interfaces instance))
                                          (effected-component m-interfaces instance)))

```



```

(failed-component m-generic-gw FAILED-NODE)
(detection m-ECC-detect Instance)
(failed-component m-generic-gw FAILED-NODE)
(detection-component m-SM DETECTION-NODE)
(correct m-lookup-backup Instance)
(failed-component m-generic-gw FAILED-NODE)
(correct-on-component m-SM Instance)
(backup-component m-backup-GW Instance)
(configuration m-core-network instance NETWORK)
(configuration m-short-term instance)
(FRAMEWORK m-short-term)
(CORRECTION-PLAN)
  (correction m-replace-with-spare Instance)
    (failed-component m-generic-gw FAILED-NODE)
    (correction-component m-crew Instance)
    (configuration m-core-network Instance)
    (backup-component m-backup-GW Instance)
    (framework m-long-term Instance)
    (effect m-failure-effect Instance)
    (sequence-of-steps 1-m-null-procedure)
    (affected-component m-crew Instance)
    (effect m-failure-effect Instance)
    (sequence-of-steps 1-m-null-procedure)
    (affected-component m-mission-support Instance)
    (effect m-failure-effect Instance)
    (sequence-of-steps 1-m-null-procedure)
    (affected-component m-systems Instance)
    (effect m-failure-effect Instance)
    (sequence-of-steps 1-m-null-procedure)
    (affected-component m-interfaces Instance)
  )

```

```

***** FANSYS: A Computer Model of Text Comprehension and Question
***** Answering for Failure Analysis
***** File: cases.sdp
***** Developed by: Sergio J. Alvarado
***** Developed by: Ronald K. Braun
***** Developed by: Kenrick J. Mock
***** Artificial Intelligence Laboratory
***** Computer Science Department
***** University of California
***** Davis, CA 95616
***** Funds for the support of this study have been allocated by the
***** NASA-Ames Research Center, Moffett Field, California, under
***** Interchange No. NCA2-721.
***** *****

(defmop 1-m-SDP-1 (m-case) instance FAILED-NODE)
  (failed-component m-generic-SDP instance FAILED-NODE)
  (mode m-operational-no-output instance)
  (failed-component m-SDP FAILED-NODE)
  (detection-node m-SDP instance DETECTION-NODE)
  (message m-generic-message instance)
  (cause m-faulty-component-cause instance)
  (failed-component m-SDP FAILED-NODE)
  (cause m-contamination-cause instance)
  (cause m-faulty-component m-SDP FAILED-NODE)
  (failed-component m-SDP FAILED-NODE)
  (cause m-temperature-out-of-range-cause instance)
  (cause m-tamper-cause instance)
  (failed-component m-SDP FAILED-NODE)
  (cause m-mechanical-shock-cause instance)
  (cause m-faulty-component m-SDP FAILED-NODE)
  (failed-component m-thermal-shock-cause instance)
  (cause m-faulty-component m-SDP FAILED-NODE)
  (failed-component m-heartbeat-detection instance)
  (detection m-heartbeat-detection instance)
  (failed-component m-SDP FAILED-NODE)
  (detection-component m-SDP FAILED-NODE)
  (correction m-look-up-backup instance)
  (failed-component m-SDP FAILED-NODE)
  (correction-component m-SM instance)
  (configuration m-core-network instance NETWORK)
  (backup-component m-backup-SDP instance)
  (framework m-short-term instance)
  CORRECTION-PLAN
  (correction m-replace-with-spare instance)
  (failed-component m-SDP FAILED-NODE)
  (correction-component m-crew instance)
  (configuration m-core-network instance)
  (backup-component m-backup-SDP instance)
  (framework m-long-term instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-crew instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-crew instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-crew instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-support instance)
  (effect m-failure-effect instance)
  (sequence-of-steps 1-m-null-procedure)
  (effected-component m-interfaces instance)
  (print 'Loading...case1)

(defmop 1-m-SDP-3 (m-case) instance FAILED-NODE)
  (failed-component m-generic-SDP instance FAILED-NODE)
  (mode m-operational-errorous-output instance)
  (failed-component m-SDP FAILED-NODE)
  (detection-node m-SM instance DETECTION-NODE)
  (message m-generic-message instance)
  (cause m-faulty-component-cause instance)
  *****
```

(detection m-undefined-detection
(failed-component m-sdp FAILED-NODE)
(sequence-of-steps m-undefined-and-group instance)
(correction m-undefined-correction
(failed-component m-sdp FAILED-NODE)
(sequence-of-steps m-undefined-and-group instance)
(framework m-short-term instance)
(correct m-replace-with-spare instance
(failed-component m-SDP FAILED-NODE)
(configuration m-core-network instance)
(backup-component m-backup-SDP instance)
(framework m-long-term instance)
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-crew instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-mission-support instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-systems instance))
(effect m-failure-effect instance
(sequence-of-steps 1-m-null-procedure)
(effected-component m-interfaces instance))

sequence-of-steps 1-m-null-procedure
(framework m-short-term instance)
(framework m-replace-with-spare instance
(failed-component m-TGU FAILED-NODE)
(correction-component m-crew instance)
(configuration m-core-network instance)
(configuration m-core-network instance)
(backup-component m-backup-TGU instance)
(framework m-long-term instance))
(effect m-failure-effect instance
sequence-of-steps 1-m-null-procedure)
(effected-component m-crew instance)
(effected-component m-crew instance)
(effect m-failure-effect instance
sequence-of-steps 1-m-null-procedure)
(framework m-mission-support instance))
(effected-component m-mission-support instance
(effect m-failure-effect instance
sequence-of-steps 1-m-null-procedure)
(effected-component m-systems instance)
(effect m-failure-effect instance
sequence-of-steps 1-m-null-procedure)
(effected-component m-interfaces instance))

)

Effectuated-component m-interfaces instance)

```

(detection-component m-SM DETECTION-NODE)
;
; Need to add switch part
; (correction m-logical-switch
;   (correction-component m-SM instance)
;     (failed-component m-network instance)
;       (backup-component m-core-network-backup instance)
;         (configuration m-DMS instance)
;           (correction m-lookup-backup instance)
;             (failed-component m-FMPAC FAILED-NODE)
;               (correction-component m-FMPAC instance)
;                 (configuration m-backup-FMPAC instance)
;                   (framework m-short-term instance)
;                     (correction m-non-operational-replacement instance
;                       (failed-component m-FMPAC FAILED-NODE)
;                         (correction-component m-crew instance)
;                           (configuration m-core-network NETWORK)
;                             (backup-component m-backup-FPAC instance)
;                               (framework m-long-term instance)
;                                 (effect m-failure-effect instance
;                                   (sequence-of-steps i-m-null-procedure)
;                                     (effected-component m-crew instance))
;                                       (effect m-failure-effect instance
;                                         (sequence-of-steps i-m-null-procedure)
;                                           (effected-component m-mission-support instance))
;                                             (effect m-failure-effect instance
;                                               (sequence-of-steps i-m-null-procedure)
;                                                 (effected-component m-systems instance))
;                                                   (effect m-failure-effect instance
;                                                     (sequence-of-steps i-m-null-procedure)
;                                                       (effected-component m-interfaces instance)))
;
```

(effected-component m-interfaces instance))

(failed-component m-GW FAILED-NODE)
(corrected-component m-SM Instance)
(corrected-component m-backup-GW Instance)
(backup-component m-backup-GW Instance)
(framework m-short-term Instance)

CORRECTION-PLAN

(correction m-non-operational-replacement Instance)
(failed-component m-MSD FAILED-NODE)
(correction-component m-crew Instance)
(configuration m-core-network NETWORK)
(backup-component m-backup-MSD Instance)
(framework m-long-term Instance)
(effect m-failure-effect Instance)
(sequence-of-steps 1-m-null-procedure)
(affected-component m-crew Instance)
(effect m-failure-effect Instance)
(sequence-of-steps 1-m-null-procedure)
(affected-component m-systems Instance)
(effect m-failure-effect Instance)
(sequence-of-steps 1-m-null-procedure)
(affected-component m-interfaces Instance))

```

*left-parent* D *right-parent* Interfaces *colon* None *period*
*eof*)
)

(setf rc.2 '(
ITEM NAME *colon* Ring Concentrator *left-paren* RC *right-parent*
FAILURE MODE *colon* Loss of output - Failure During Operation
FAILURE CAUSES *colon* Piece-part failures *comma* Contamination
*comma* Temperature *left-paren* High or Low *right-paren* *comma*
Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *slash* VERIFICATION *colon* Indication of a RC
*quote* loss of output *quote* during operations is first detected by
the *quote* next *quote* active node on the network *period* Local
System Management *left-paren* SM *right-parent* SM *quote* within the *quote*
next *quote* node will reach a time-out limit for receipt of the
network token *period*

CORRECTIVE ACTION *colon*
*****  

*left-parent* A *right-parent* Short Term *colon* OMS will automatically
reconfigure the network and
present information regarding the reconfigured network to crew
and ground controllers *period* Corrective action options consist
of *left-parent* 1 *right-parent* Selectover of all communications
*slash* support to the backup network *comma*
*left-parent* 2 *right-parent* bypass of only the failed RC on
the primary network using the backup network *period*
*left-parent* B *right-parent* Long Term *colon* Failed RC ORUs are
removed and replaced via a logistics spare *period*
FAILURE EFFECT ON *colon*
*left-parent* A *right-parent* Crew *slash* SSPE *colon* None *period*
The conditions associated with the replacement
of a RC within the network configuration are such that the network
is already in a reconfigured state supplying full services *period*
*left-parent* B *right-parent* Mission Support *colon* None *period*
*left-parent* C *right-parent* System *colon* None *period*
*left-parent* D *right-parent* Interfaces *colon* None *period*
ITEM NAME *colon* Ring Concentrator *left-paren* RC *right-parent*
*eof*)

(setf rc.3 '(
ITEM NAME *colon* A Computer Model of Text Comprehension and Question
Answering for Failure Analysis
File: cases.txt
Developed by: Sergio J. Alvarado
Ronald K. Brun
Kenrick J. Mock
Artificial Intelligence Laboratory
Computer Science Department
University of California
Davis, CA 95616
Funds for this study have been allocated by the
NASA-Ames Research Center, Moffett Field, California, under
Interchange No. NCA2-721.
*****  

(setf rc.1 '(
ITEM NAME *colon* Ring Concentrator *left-paren* RC *right-parent*
FAILURE MODE *colon* Loss of output - Failure to Start
FAILURE CAUSES *colon* Piece-part failures *comma* Contamination
*comma* Temperature *left-paren* High or Low *right-parent* *comma*
Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *slash* VERIFICATION *colon* Indication of a RC
*quote* failure to start *quote* is first detected by the *quote*
next *quote* active node on the network *period* Local System
Management *left-paren* SM *right-parent* Within the *quote* next
*quote* node will reach a time-out limit for receipt of the network
token *period*

CORRECTIVE ACTION *colon*
*left-parent* A *right-parent* Short Term *colon* Network
reconfiguration is effected automatically *period* The DMS network
remains in operation in a reconfigured state with the failed RC
bypassed *period*
*left-parent* B *right-parent* Long Term *colon* Failed RC that
fails to become operable *comma* the crewmen check for *quote* applied
power *quote* and check for *quote* connector tightness *quote*
*period* If the RC cannot then be placed in operation *comma* it is
removed and replaced with an ORU logistics spare *period*
FAILURE EFFECT ON *colon*
*left-parent* A *right-parent* Crew *slash* SSPE *colon* None *period*
The conditions associated with the replacement
of a RC within the network configuration are such that the network
is already in a reconfigured state supplying full services
*period*
*left-parent* B *right-parent* Mission Support *colon* None *period*
*left-parent* C *right-parent* System *colon* None *period*
FAILURE MODE *colon* Erroneous Output *left-paren* Transmitter Stuck
High *slash* Low *right-parent* - Failure During Operation
FAILURE CAUSES *colon* Piece-part failures *comma* Contamination
*comma* Temperature *left-paren* High or Low *right-parent* *comma*
Mechanical shock *comma* Thermal Shock
FAILURE DETECTION *slash* VERIFICATION *colon* Indication of a RC
stuck at high *slash* low failure is first detected by the *quote*
```

Independent of the funding period*

Management * left-parent* SM *right-parent* Within the *quote* next *quote* nodes will reach a time-out limit for receipt of the network token * period*

- CORRECTIVE ACTION *colon***
 - *left-paren* A *right-paren* Short Term *colon* OMS will automatically reconfigure the network and present information regarding the reconfigured network to crew and ground controllers *period* Corrective action options consist of *left-paren* 1 *right-paren* selector of all communications *slash* support to the backup network *comma*
 - *left-paren* 2 *right-paren* by-pass of only the failed RC or *left-paren* 2 *right-paren* using the backup network *period*

left-paren B *right-paren* Long Term *colon* Failed RC ORUS are removed and replaced via a logistics spare *period*

FAILURE EFFECT ON SERVER

* left-parent* A *right-parent* Crew *slash* SSPE *colon* None *period*

The conditions associated with the replacement of a RC within the network configuration are such that the network is already in a reconfigured state supplying full services

period

left-paren B *right-paren* Mission Support *colon* None *period*
left-paren C *right-paren* System *colon* None *period*
left-paren D *right-paren* Interfaces *colon* None *period*

```
* EOF*)
```

```
(setf gw:1 '(
```

```
    :left-parent :right-parent*))
```

ITEM NAME *colon* Gateway *port=serial
FAILURE MODE *colon* Loss of output - Failure to Start

FAILURE CAUSES *colon* *success* *slash* *right-parent* *comm
comma *Temperature* *left-parent* High or Low *right-parent* *comm
Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *slash* VERIFICATION *colon* Detection *slash*
Verification of failure to start will occur through the use of
period *colon* *right-parent* *period*

On Self Test *left-paren* OMA *slash* SM
CORRECTIVE ACTION *colon*
left-paren A *right-paren* Short Term *colon* OMA *slash* SM
software directs PMAD to power-off the failed GW *period*. A
look-up table maintained within DMS selects the redundant GW
to recover from the replacement.

period OMA *slash* SM then directs PMAD to power on. Once on:
left-paren B *right-paren* Long Term *colon* Failed GWS are
left-paren IVA *right-paren* and replaced via
transport to *slash* from the ground *period*
period

FAILURE EFFECT ON *colon*

Independent of the functioning of the GWS *period*
left-parent B *right-parent* Mission Support *colon* None *period* The
JEM network can function autonomously as can the ESM network and the DMS network *period*

* left-paren* C *right-paren* System *colon* None *period* The operation of the DNS network is independent of the operation of the GWs *period*

eoc)

ITEM NAME *colon* Gateway *left-paren* GW *right-paren*
FAILURE MODE *colon* Loss of output - Failure during operation

FAILURE CAUSES *colon* Piece-part failures *comma* Contamination
comma Temperature *left-paren* High or Low *right-paren* *
Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *slash* **VERIFICATION** *colon* **Failure detection** verification of the GW loss of output will occur via a node verification of the DMS System Management function of the DMS System Management function of this function. **Status data management** *left-paren* **SM** *right-**paren*** **period**. A subtask of this function.

The sending of periodic "quote" heartbeat messages "quote" attached to the DMS Ring Concentrators "left-para" RCS * rig to verify its operational status * period. Failure of one to respond to the heartbeat message is indication of an ORU power output failure * period*

left-paren B *right-parent* Long Term *colon* Failed GWS as
left-paren IVA *right-parent* and replaced via
transport to *slash* from the ground *period*

*FAILURE EFFECT ON *colon*
left-paren A *right-paren* Crew *slash* SSPE *colon* Noncolon*
Basic station and crew support functions are
complementing of the GNS *period*

independent of the functioning of the network. The colon character is used to separate the payload from the address information. The payload can function autonomously as can the core network period.

left-parent C *right-parent* System *colon* None *period*
autonomous
operation of the GMs network is independent of the
operation of the GMs *period*

left-paren D *right-paren* Interfaces *colon* None *per-selectover to the redundant GW is effected *period*

left-paren A *right-paren* Short Term *colon* OMA *slash* SM software directs PMAD to power-off the failed processor *period* A look-up table maintained within DMS selects a suitable replacement processor *period* OMA *slash* SM then directs PMAD to power-on a replacement processor *period*

ITEM NAME *colon* Gateway *left-paren* GW *right-paren*

FAILURE MODE *colon* Erroneous NIA Output *left-paren* DMS side of the Interface *right-paren* - Failure during operation

FAILURE CAUSES *colon* Piece-part failures *comma* Erroneous Input

CORRECTIVE ACTION *colon*

FAILURE DETECTION *slash* VERIFICATION *colon* Data packet *left-paren* A *right-paren* Short Term *colon* OMA *slash* SM transmission across the DMS network will include Error Correction Code ECC *right-paren* *period* Inclusion of ECC will allow nodes receiving data packets to perform single bit error detection and correction *comma* and two bit error detection *period*

left-paren B *right-paren* Mission Support *colon* None *period*

left-paren C *right-paren* Crew *colon* SSPE *colon* None *period*

left-paren D *right-paren* System *colon* None *period* In general the SDP-4B processors support applications in pairs with the first unit active and the second in a cold backup status *period*

left-paren E *right-paren* Interfaces *colon* None *period*

left-paren F *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren G *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren H *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren I *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren J *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren K *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren L *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren M *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren N *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren O *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren P *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren Q *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren R *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren S *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren T *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren U *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren V *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren W *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren X *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren Y *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

left-paren Z *right-paren* Selectover to the redundant SDP-4B occurs for any application requiring it *period*

FAILURE EFFECT ON *colon*

left-paren A *right-paren* Short Term *colon* OMA *slash* SM software directs PMAD to power-off the failed GW *period* A look-up table maintained within DMS selects the redundant GW *period* OMA *slash* SM then directs PMAD to power-on the replacement GW *period*

left-paren B *right-paren* Failed GWs are removed *left-paren* C *right-paren* and replaced via transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*

left-paren A *right-paren* Long Term *colon* OMA *slash* SM The probability of an output device responding to an erroneous output is considered negligible *period*

left-paren B *right-paren* Mission Support *colon* None *period* The payload network can function autonomously as can the core network *period*

left-paren C *right-paren* System *colon* The operation of the DMS network is independent of the operation of the GWs *period*

left-paren D *right-paren* Interfaces *colon* None *period* Selectover to the redundant GW is effected *period*

left-paren E *right-paren* Verification *colon* OMA *slash* SM *left-paren* F *right-paren* High or Low *right-paren* *comma* Contamination *comma* Temperature *left-paren* High or Low *right-paren* *comma* Mechanical shock *colon* Thermal Shock

left-paren G *right-paren* Verification *colon* OMA *slash* SM *left-paren* H *right-paren* Selectover to each ORU attached to the DMS *period* A subtask of this function is the sending of periodic *period* heartbeat messages *quote* to each ORU attached to the DMS Ring Concentrators *left-paren* RCs *right-paren* to verify its operational status *period* Failure of any of these ORUs to respond to TAD consecutive heartbeat message is indication of an ORU loss of power output failure *period*

CORRECTIVE ACTION *colon*

left-paren A *right-paren* Short Term *colon* OMA *slash* SM software directs PMAD to power-off the failed processor *period* A look-up table maintained within DMS selects a suitable replacement processor *period* OMA *slash* SM then directs PMAD to power-on a replacement processor *period*

FAILURE MODE *colon* Loss of output - Failure to Start

FAILURE CAUSES *colon* Piece-part failures *comma* Contamination *comma* Temperature *left-paren* High or Low *right-paren* *comma* Mechanical shock *colon* Thermal Shock

FAILURE DETECTION *slash* VERIFICATION *colon* Detection *slash* Verification of failure to start will occur through the use of a Power On Self Test *left-paren* POST *right-paren* *period*

left-paren A *right-paren* Crew *slash* SSPE *colon* None *period*
 Detected erroneous transmissions will be
 discarded resulting in no impact to crew *slash* SSPE *period*

left-paren B *right-paren* Mission Support *colon* None *period*
 Selectover to a redundant SDF-4B occurs for
 any application requiring it *period*

left-paren C *right-paren* System *colon* None *period* In general
 the SDF-4B processors support applications
 in pairs with the first unit active and the second in a cold
 backup status *period*

left-paren D *right-paren* Interfaces *colon* None *period*
 Selectover to the redundant SDF-4B occurs for
 any application requiring it *period*

eoc)

(setf tgu.1 '(
 ITEM NAME *colon* Time Generation Unit *left-paren* TGU *right-paren*
 FAILURE MODE *colon* Loss of output - Failure to Start
)

FAILURE CAUSES *colon* Piece-part failures *comma* Contamination
 comma Temperature *left-paren* High or Low *right-paren* *comma*
 Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *slash* VERIFICATION *colon* Detection *slash*
 Verification of failure to start will occur through the use of a Power
 On Self Test *left-paren* POST *right-paren* *period*

CORRECTIVE ACTION *colon*

left-paren A *right-paren* Short Term *colon* TBA

left-paren B *right-paren* Long Term *colon* Failed TGU's are removed
 left-paren C *right-paren* System *colon* None *period* The
 operation of the DMS network is independent of the
 operation of the TGU *period*

FAILURE EFFECT ON *colon*

left-paren A *right-paren* Crew *slash* SSPE *colon* None *period*
 left-paren B *right-paren* Mission Support *colon* TBA

left-paren C *right-paren* IVA *colon* None *period* The
 short term corrective actions will consist of immediate selectover to
 the Local Bus *period* If it has been determined by SM *slash* OMA
 that BCU *slash* BIA have failed
 and low priority operations are effective *comma* then corrective
 actions will consist of SDF-4B replacement or reassignment of the
 resident function to a cold backup *period*

If it has been determined by SM *slash* OMA that BCU *slash* BIA
 have failed and the SDF-4B is performing high priority functions *comma*
 a short term corrective actions will consist of immediate selectover to
 a warm backup *period* If it has been determined by SM *slash* OMA
 that BCU *slash* BIA have failed
 and low priority operations are effective *comma* then corrective
 actions will consist of SDF-4B replacement or reassignment of the
 resident function to a cold backup *period*

left-paren B *right-paren* Long Term *colon* Failed SDF-4B ORUS are
 removed *left-paren* IVA *right-paren* and replaced via
 transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*

ITEM NAME *colon* Fixed Multi-Purpose Applications Console
 left-paren F-MPAC *right-paren* *left-paren* Processor *right-paren*

FAILURE MODE *colon* Loss of output - Failure to Start

FAILURE CAUSES *colon* Piece-Part failures *comma* Contamination
 comma Temperature *left-paren* High or Low *right-paren* *comma*
 Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *slash* VERIFICATION *colon* Detection *slash*
 Verification of failure to start will occur through the use of a Power
 On Self Test *left-paren* POST *right-paren* *period*

CORRECTIVE ACTION *colon*

left-paren A *right-paren* Short Term *colon* OMA *slash* SM
 left-paren A *right-paren* Short Term *colon* OMA *slash* SSPE *colon* None *period*
 software directs PMAD to power-off the failed F-MPAC *period*
 A look-up table maintained within DMS selects the replacement
 F-MPAC *period*. OMA *slash* SM then directs PMAD to power-on the
 replacement F-MPAC *period*

left-paren B *right-paren* Long Term *colon* Failed F-MPACs are
 removed *left-paren* IVA *right-paren* and replaced via
 transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*

left-paren A *right-paren* Crew *slash* SSPE *colon* None *period*
 left-paren B *right-paren* Mission Support *colon* TBA

left-paren C *right-paren* System *colon* None *period* The
 operation of the DMS network is independent of the
 operation of the TGU *period*

left-paren D *right-paren* Interfaces *colon* TBA

left-paren E *right-paren* Time Generation Unit *colon* None *period*

FAILURE MODE *colon* Erroneous Output - Failure during operation

FAILURE CAUSES *colon* Piece-Part failures *comma* Erroneous input

FAILURE DETECTION *slash* VERIFICATION *colon* To be supplied *period*

CORRECTIVE ACTION *colon*

left-paren A *right-paren* Short Term *colon* TBA

left-paren B *right-paren* Mission Support *colon* TBA

left-paren C *right-paren* System *colon* None *period* The
 operation of the DMS network is independent of the
 operation of the TGU *period*

FAILURE EFFECT ON *colon*

left-paren A *right-paren* Crew *slash* SSPE *colon* None *period*
 left-paren B *right-paren* Mission Support *colon* TBA

left-paren C *right-paren* System *colon* None *period* The
 operation of the DMS network is independent of the
 operation of the TGU *period*

FAILURE DETECTION *colon* Verification *colon* Failure detection and
 verification of the loss of output will occur via a node health and
 status data management function of the DMS System Management software
 period A subtask of this function is the sending of periodic
 quote heartbeat messages *quote* to each ORU attached to the DMS
 Ring Concentrators *left-paren* RCS *right-paren* to verify its
 operational status *period* Failure of any of these nodes to respond
 to the heartbeat message is indication of a nodal loss of output
 period

FAILURE DETECTION *slash* VERIFICATION *colon* Indications of loss of
 output and self test capabilities occur via presence of the GPS input
 signal *comma* quality of the GPS input signal *comma* loss of
 internal clock *comma* quality of internal clock signal *comma*
 detection of local bus errors *comma* detection of memory errors
 left-paren parity *comma* ECC *right-paren* and Time Distribution
 Bus *left-paren* TBD *right-paren* driver Failure *period*

CORRECTIVE ACTION *colon*

left-paren A *right-paren* Short Term *colon* TBA

left-paren B *right-paren* Long Term *colon* Failed TGUs are removed
 left-paren IVA *right-paren* and replaced via
 transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*

left-paren A *right-paren* Crew *slash* SSPE *colon* None *period*
 left-paren B *right-paren* Mission Support *colon* None *period* The
 crew can use a redundant F-MPAC *period*

left-paren C *right-paren* System *colon* None *period* The
 operation of the DMS network is independent of the
 operation of the F-MPAC processors *period*

left-paren D *right-paren* Interfaces *colon* None *period* The crew
 can use a redundant F-MPAC *period*

eoc)

(setf fmpac.2 (

(setf tgu.3 (

ITEM NAME *colon* Fixed Multi-Purpose Applications Console
 left-paren F-MPAC *right-paren* *left-paren* Processor *right-paren*

FAILURE MODE *colon* Loss of output - Failure during operation

FAILURE CAUSES *colon* Piece-part failures *comma* Contamination
 comma Temperature *left-paren* High or Low *right-paren* *comma*
 Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *colon* VERIFICATION *colon* Failure detection and
 verification of the loss of output will occur via a node health and
 status data management function of the DMS System Management software
 period A subtask of this function is the sending of periodic
 quote heartbeat messages *quote* to each ORU attached to the DMS
 Ring Concentrators *left-paren* RCS *right-paren* to verify its
 operational status *period* Failure of any of these nodes to respond
 to the heartbeat message is indication of a nodal loss of output
 period

eoc))

Selectover to a redundant F-MPAC processor

CORRECTIVE ACTION *colon*

left-parent A *right-parent* Short Term *colon* OMA *slash* SM software directs PMAD to power-off the failed F-MPAC *period* A look-up table maintained within DMS selects the replacement F-MPAC *period* OMA *slash* SM then directs PMAD to power-on the replacement F-MPAC *period*

left-parent B *right-parent* Long Term *colon* Failed F-MPACs are removed *left-parent* IVA *right-parent* and replaced via transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*

left-parent A *right-parent* Crew *slash* SSE *colon* None *period* The crew selects a redundant F-MPAC processor *period*

left-parent B *right-parent* Mission Support *colon* None *period* Selectover to a redundant F-MPAC processor occurs *period*

left-parent C *right-parent* System *colon* None *period* The operation of the DMS network is independent of the operation of the F-MPAC processors *period*

left-parent D *right-parent* Interfaces *colon* None *period* Selectover to a redundant F-MPAC occurs *period*

eoc)

(setf fmpac.3 ' {

ITEM NAME *colon* Fixed Multi-Purpose Applications Console *left-parent* F-MPAC *right-parent* *left-parent* Processor *right-parent* *left-parent* F-MPAC *right-parent* *left-parent* Erroneous NIA Output - Failure during operation

FAILURE MODE *colon* Data Packet transmission across the DMS network will include Error Correction Code *left-parent* ECC *right-parent* *period* Inclusion of ECC will allow nodes receiving data packets to perform single bit error detection and correction *comma* and two bit error detection *period*

CORRECTIVE ACTION *colon*

left-parent A *right-parent* Short Term *colon* Two possible corrective actions are *left-parent* 1 *right-parent* selectover to the backup network *comma* and *left-parent* 2 *right-parent* selectover to a warm or cold backup F-MPAC processor *period*

left-parent B *right-parent* Long Term *colon* Failed F-MPACs are removed *left-parent* IVA *right-parent* and replaced via transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*

left-parent A *right-parent* Crew *slash* SSE *colon* None *period* SM *slash* OMA will direct the processor to continue operations using the backup network communications *period*

occurs *period*

left-parent C *right-parent* System *colon* None *period* The operation of the DMS network is independent of the operation of the F-MPAC processors *period*

left-parent D *right-parent* Interfaces *colon* None *period* can use a redundant F-MPAC *period*

eoc)

(setf msd.1 ' {

ITEM NAME *colon* Mass Storage Device *left-parent* MSD *right-parent*

FAILURE MODE *colon* Loss of output - Failure to Start

FAILURE CAUSES *colon* Piece-part failures *comma* Contamination *comma* Temperature *left-parent* High or Low *right-parent* *comma* Mechanical shock *comma* Thermal Shock

Failure detection *slash* verification *colon* To be supplied *period*

CORRECTIVE ACTION *colon*

left-parent A *right-parent* Short Term *colon* TBA

left-parent B *right-parent* Long Term *colon* Failed MSDs are removed *left-parent* IVA *right-parent* and replaced via transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*

left-parent A *right-parent* Crew *slash* SSE *colon* None *period* The OMA *slash* SM selects a redundant MSD *period*

left-parent B *right-parent* Mission Support *colon* None *period* Selectover to a redundant MSD occurs *period*

left-parent C *right-parent* System *colon* None *period* *left-parent* D *right-parent* Interfaces *colon* None *period* *left-parent* E *right-parent* Selectover to a redundant MSD occurs for any application requiring it *period*

eoc)

(setf msd.2 ' {

ITEM NAME *colon* Mass Storage Device *left-parent* MSD *right-parent*

FAILURE MODE *colon* Loss of output - Failure during operation

FAILURE CAUSES *colon* Piece-part failures *comma* Contamination *comma* Temperature *left-parent* High or Low *right-parent* *comma* Mechanical shock *comma* Thermal Shock

Failure detection *slash* verification *colon* To be supplied *period*

CORRECTIVE ACTION *colon*

left-parent A *right-parent* Short Term *colon* TBA

left-parent B *right-parent* OMA will direct the processor to continue operations using the backup network communications *period*

left-parent B *right-parent* Long Term *colon* Failed MSDs are removed
 left-parent IVA *right-parent* and replaced via
 transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*
 left-parent A *right-parent* Crew *slash* SSPE *colon* None *period*
 The OMA *slash* SM can use a redundant F-MPAC with
 its MSD if required *period*

FAILURE MODE *colon*
 left-parent B *right-parent* Mission Support *colon* None *period* The
 crew can select a redundant F-MPAC
 with MSD for any application requiring it *period*

left-parent* C *right-parent* System *colon* None *period
 left-parent D *right-parent* Interfaces *colon* None *period*
 Selector over to a redundant MSD occurs for any
 application requiring it *period*

***eoc*)**
 (setf msd.3 ' (ITEM NAME *colon* Mass Storage Device *left-parent* MSD *right-parent*
 ITEM NAME *colon* Cupola Multi-Purpose Applications Console

FAILURE MODE *colon*
 left-parent C *right-parent* Erroneous Output - Missing Data During operation

FAILURE CAUSES *colon*
 Piece-part failures

FAILURE DETECTION *colon*
 VERIFICATION *colon* To be supplied *period*

CORRECTIVE ACTION *colon*
 left-parent A *right-parent* Short Term *colon* TBA

left-parent* B *right-parent* Long Term *colon Failed MSDs are removed
 left-parent IVA *right-parent* and replaced via
 transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*
 left-parent A *right-parent* Crew *slash* SSPE *colon* None *period*
 The OMA *slash* SM can use a redundant F-MPAC with its
 MSD if required *period*

left-parent* B *right-parent* Mission Support *colon None *period* The
 crew can select a redundant F-MPAC
 with MSD for any application requiring it *period*

left-parent* C *right-parent* System *colon None *period*
 left-parent D *right-parent* Interfaces *colon* None *period* The crew
 selects a redundant F-MPAC with its
 attached MSD *period*

FAILURE CAUSES *colon* Piece-part failures *comma* Contamination
 comma Temperature *left-parent* High or Low *right-parent* *period*
 Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *colon* VERIFICATION *colon* Failure detection and
 Verification of the loss of output will occur via a node health and
 status data management function of the DMS System Management
 left-parent SM *right-parent* software *period* A subtask of this
 function is the sending of periodic *quote* heartbeat messages *quote*
 to each ORU attached to the DMS Ring Concentrators *left-parent* RGS
 right-parent to verify its operational status *period* Failure of any
 of these nodes to respond to TBD consecutive heartbeat message is

CORRECTIVE ACTION *colon*
 left-parent A *right-parent* Short Term *colon* DMS SM and OMA
 software directs PMAD to power-off the
 failed C-MPAC processor *period* The crew can then replace this
 processor with a logistics spare and resume the startup operation *period*

left-parent* B *right-parent* Long Term *colon Failed C-MPAC
 processors are removed *left-parent* IVA *right-parent* and replaced via
 transport to *slash* from the ground *period*

FAILURE EFFECT ON *colon*
 left-parent A *right-parent* Crew *slash* SSPE *colon* None *period* The
 crew can selector over to the redundant C-MPAC
 should one fail to start *period* In addition *comma* for certain
 applications that are not limited to hemispherical viewing from the cupola
 comma the crew can select instead a F-MPAC *period*

left-parent* B *right-parent* Mission Support *colon None *period* The
 crew can selector over to the redundant
 C-MPAC should one fail to start *period* In addition *comma* for certain
 applications that are not limited to hemispherical viewing from the cupola
 comma the crew can select instead a F-MPAC *period*

left-parent* C *right-parent* System *colon None *period* The
 operation of the DMS network is independent of the
 operation of the C-MPAC processors *period*

left-parent* D *right-parent* Interfaces *colon None *period* The crew
 can selector over to the redundant
 C-MPAC should one fail to start *period*

***eoc*)**
 (setf cmpac.2 ' (ITEM NAME *colon* Cupola Multi-Purpose Applications Console

FAILURE MODE *colon*
 Loss of output - Failure during operation

FAILURE CAUSES *colon*
 Piece-part failures *comma* Contamination
 comma Temperature *left-parent* High or Low *right-parent* *comma*
 Mechanical shock *comma* Thermal Shock

FAILURE DETECTION *colon*
 VERIFICATION *colon* Failure detection and
 verification of the loss of output will occur via a node health and
 status data management function of the DMS System Management

transport to 'slash' from the ground *period*

FALLURE EFFECT ON *colon*

left-paren A *right-parent* Short Term *colon* For operations requiring a hemispherical viewing capability *comma* the crew must determine whether or not these operations require this viewing from the particular cupola housing the failed C-MPAC processor *period* If viewing from that cupola is essential *comma* then no redundancy exists and replacement of the failed C-MPAC processor cupola *period*

For operations not requiring a hemispherical viewing capability *comma* a look-up table maintained within DMS selects a suitable replacement F-MPAC workstation to perform the function of the C-MPAC workstation *period* DMS SM then directs PMAD to power-on the replacement processor *period*

left-paren C *right-parent* System *colon* None *period* The operation of the DMS network is independent of the operation of the C-MPAC processors *period*

left-paren D *right-parent* Interfaces *colon* None *period* The DMS network continues to function despite the failure to function of the C-MPAC *period*

ecc)

(setf cmpac.3 ' (

ITEM NAME *colon* Cupola Multi-Purpose Applications Console

left-paren C-MPAC *right-parent* Processor

FAILURE MODE *colon* Erroneous NIA Output - Failure during operation

FAILURE CAUSES *colon* Piece-part failures

FAILURE DETECTION *slash* VERIFICATION *colon* Data packet transmission across the network will include Error Correction Code

left-paren ECC *right-parent* *period* Inclusion of ECC will allow nodes receiving data packets to perform single bit error detection and correction *comma* and two bit error detection *period*

CORRECTIVE ACTION *colon*

left-paren A *right-parent* Short Term *colon* For operations requiring a hemispherical viewing capability *comma* the crew must first determine whether or not these operations require this viewing from the particular cupola housing the failed C-MPAC processor *period* If viewing from that cupola is essential *comma* then no redundancy exists and replacement of the failed C-MPAC processor must take place before operations can resume *period* If viewing and operations can be accomplished instead from the other cupola *comma* then appropriate notification is made to all command and control operations personnel *comma* and operations are conducted from the other cupola *period*

For operations not requiring a hemispherical viewing capability *comma* a look-up table maintained within DMS selects a suitable replacement F-MPAC workstation to perform the function of the C-MPAC workstation *period* DMS SM then directs PMAD to power-on the replacement processor *period*

left-paren B *right-parent* Long Term *colon* Failed C-MPAC